



IBM Research



The BlueGene/L Supercomputer: Delivering Large Scale Parallelism



José E. Moreira

IBM T. J. Watson Research Center

The Blue Gene Project – A Little History

- In December 1999, IBM Research announced a five-year, US\$100 million, effort to build a petaflops scale supercomputer to attack problems such as protein folding
- The Blue Gene project has two primary goals:
 - ❖ Advance the state of the art in biomolecular simulations
 - ❖ Advance the state of the art in computer design and software for extremely large scale systems
- In November 2001, a research partnership with Lawrence Livermore National Laboratory was announced
- In November 2002, a <<\$100 M US acquisition of a BG/L machine by LLNL from IBM as part of the ASCI Purple contract was announced
- And here we are!

Outline

- BlueGene/L high-level design philosophy
- BlueGene/L system architecture and technology overview
- BlueGene/L software architecture
- BlueGene/L in operation
- BlueGene/L programming

Outline

- BlueGene/L high-level design philosophy
- BlueGene/L system architecture and technology overview
- BlueGene/L software architecture
- BlueGene/L in operation
- BlueGene/L programming

BlueGene/L Philosophy

- Some applications display very high levels of parallelism: they can be executed efficiently on tens of thousands of processors if:
 - ❖ low-latency, high-bandwidth interconnect is present
 - ❖ machine is reliable enough
 - ❖ good compilers and programming models are available
 - ❖ machine is manageable (simple to build and operate)
- Vastly improved price-performance for a class of applications by choosing simple low-power building block - highest possible single-threaded performance is not relevant, aggregate is!
- Scalable architecture and appropriate package can lead to a high density, low power, massively parallel system
- Cellular architecture + aggressive packaging + scalable software = BlueGene/L

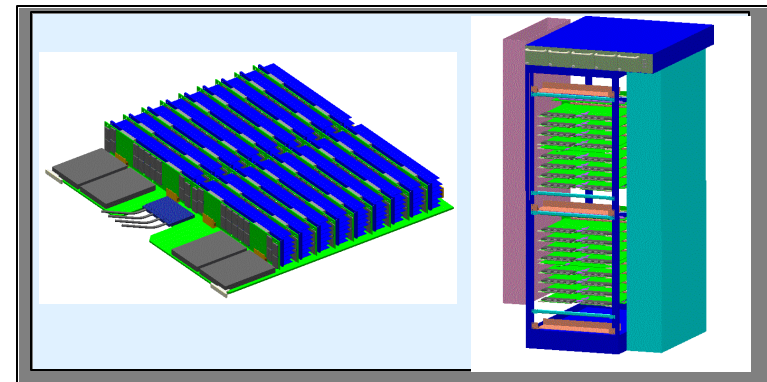
Compute Density



**2 Pentium 4/1U
42 1U/rack
84 processors/rack**



**2 Pentium 4/blade (7U)
14 blade/chassis
6 chassis/frame
168 processors/rack**



**2 dual CPU/compute card
16 compute cards/node card
16 node cards/midplane
2 midplanes/rack
2048 processors/rack**

The Problem: A Hardware Perspective

- The current approach to large systems is to build clusters of large SMPs (NEC Earth Simulator, ASCI machines, Linux clusters)
 - ❖ SMPs are typically designed for something else (sweet spot of market) and then combined “unnaturally”
 - ❖ Very expensive switches for high performance
 - ❖ Very high electrical power consumption: low computing power density
 - ❖ Significant amount of resources (particularly in memory hierarchy) devoted to improving single-thread performance
- Would like a more modular/cellular approach, with a simple building block (or cell) that can be replicated *ad infinitum* as necessary – aggregate performance is important
- Cell should have low power/high density characteristics, and should also be cheap to build and easy to interconnect

Approach: cellular system architecture

- A homogeneous collection of simple independent processing units called cells, each with its own operating system image
- All cells have the same computational and communications capabilities (interchangeable from OS or application view)
- Integrated connection hardware provides a straightforward path to scalable systems with thousands/millions of cells
- Our goal with BlueGene/L is a system with 64k compute nodes
- Challenges:
 - ❖ programmability (particularly for performance)
 - ❖ system management
 - ❖ fault-tolerance and high-availability
 - ❖ mapping of computations to cells

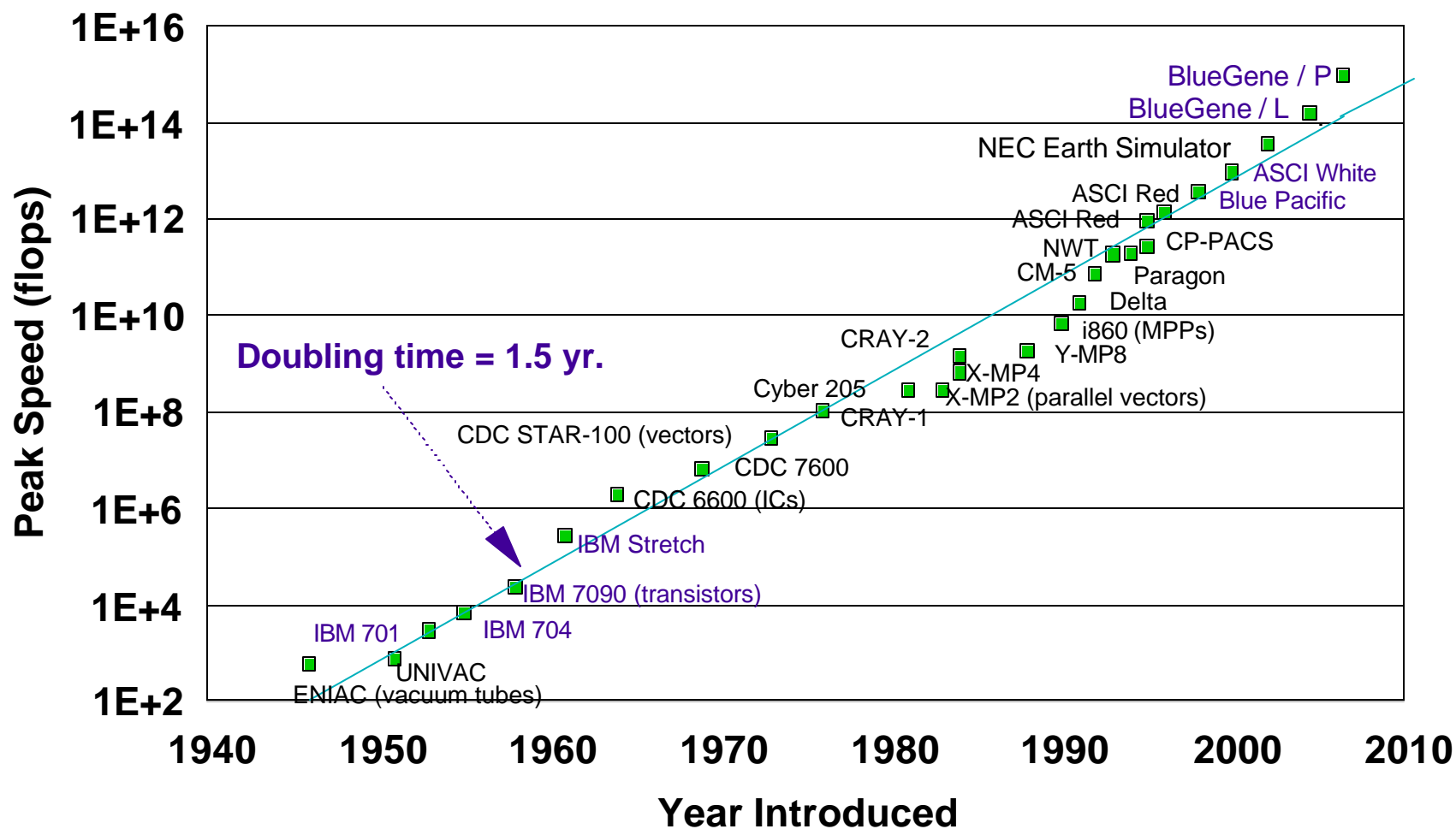
The Problem: A Software Perspective

- Limited success with systems of ~1,000 nodes
 - ❖ System management scalability issues
 - ❖ System reliability
 - ❖ Application performance scalability
- Application scalability depends heavily on specific application, but there are known applications that can use > 10,000 nodes
 - ❖ High-bandwidth, low-latency interconnect necessary
 - ❖ Minimum operating system involvement
- Most system management tools (e.g., job scheduling systems, debuggers, parallel shells) are unlikely to work well with 10,000 – 100,000 nodes
 - ❖ It is unlikely that everything will be up at the same time
 - ❖ Time-outs, stampedes, other problems

Approach: hierarchical system software

- From a system perspective, BlueGene/L will look like a 1024-way cluster of independent machines - a manageable size
- Each machine is under control of one Linux image
- Each machine has 64 compute nodes attached to it, which can be used exclusively for running user application processes
- 1000-way cluster organization can leverage existing cluster infrastructure for single-system image

Supercomputer Peak Speed

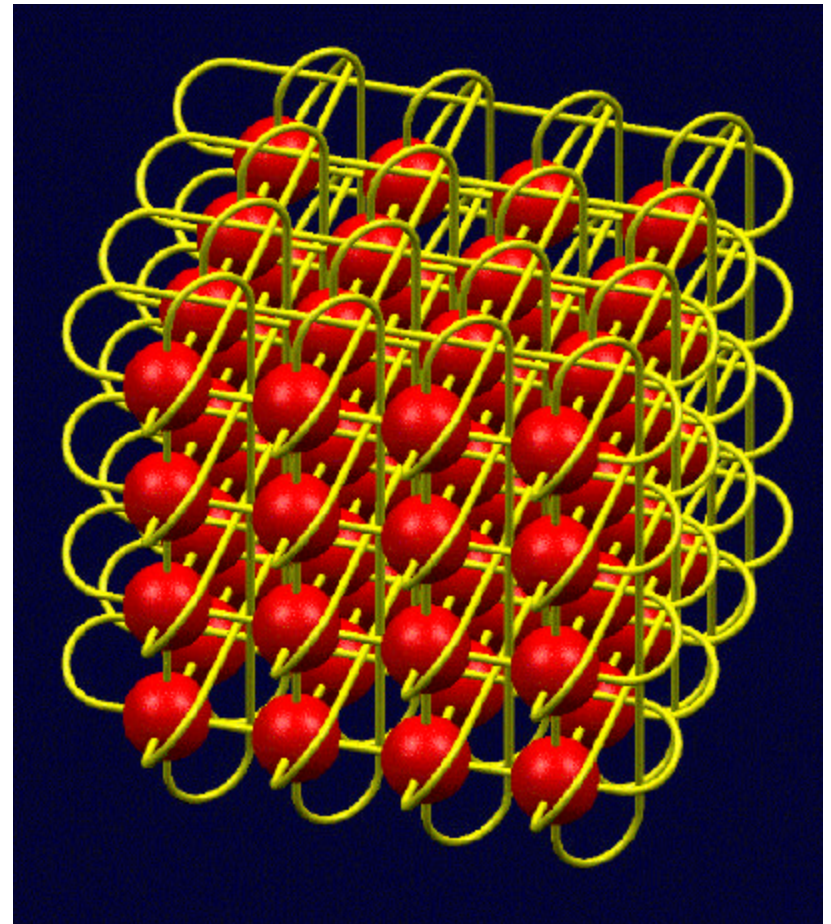


Outline

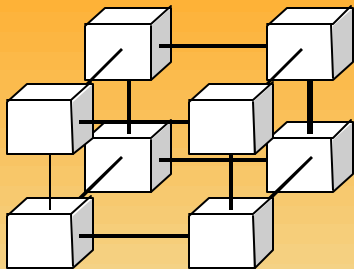
- BlueGene/L high-level design philosophy
- **BlueGene/L system architecture and technology overview**
- BlueGene/L software architecture
- BlueGene/L in operation
- BlueGene/L programming

BlueGene/L Fundamentals

- A large number of nodes (65,536)
 - ❖ Low-power (20W) nodes for density
 - ❖ High floating-point performance
 - ❖ System-on-a-chip technology
- Nodes interconnected as 64x32x32 three-dimensional torus
 - ❖ Easy to build large systems, as each node connects only to six nearest neighbors – full routing in hardware
 - ❖ Bisection bandwidth per node is proportional to n^2/n^3
 - ❖ Auxiliary networks for I/O and global operations
- Applications consist of multiple processes with message passing
 - ❖ Strictly one process/node
 - ❖ Minimum OS involvement and overhead



BlueGene/L Interconnection Networks



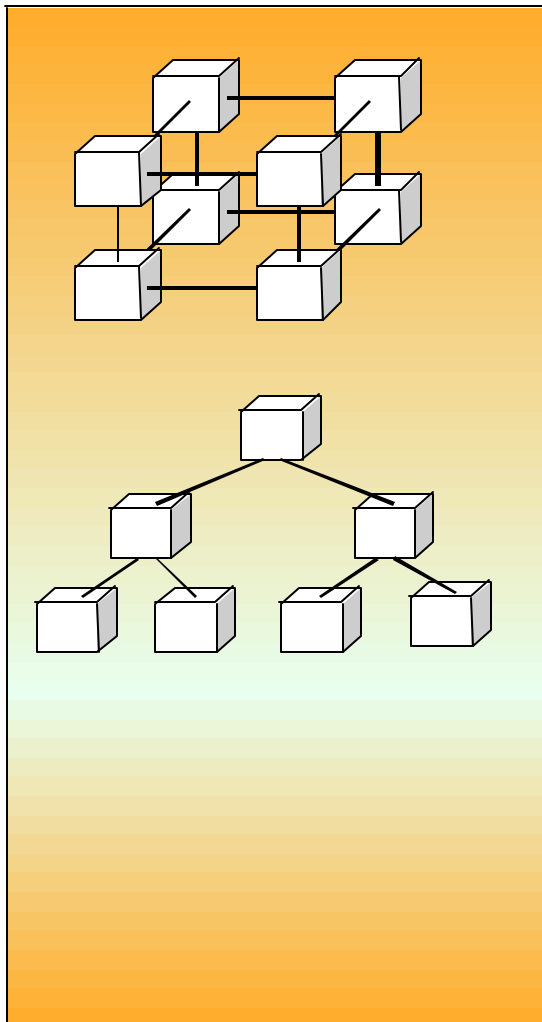
3 Dimensional Torus

- ❖ Interconnects all compute nodes (65,536)
- ❖ Virtual cut-through hardware routing
- ❖ 1.4Gb/s on all 12 node links (2.1 GB/s per node)
- ❖ Communications backbone for computations
- ❖ 350/700 GB/s bisection bandwidth

BlueGene/L Fundamentals (continued)

- Machine should be dedicated to execution of applications, not system management
 - ❖ Avoid asynchronous events (e.g., daemons, interrupts)
 - ❖ Avoid complex operations on compute nodes
- The “I/O node” – an offload engine
 - ❖ System management functions are performed in a (N+1)th node
 - ❖ I/O (and other complex operations) are shipped from compute node to I/O node for execution
 - ❖ Number of I/O nodes adjustable to needs (N=64 for BG/L)
- This separation between application and system functions allows compute nodes to focus on application execution
- Communication to the I/O nodes must be through a separate tree interconnection network to avoid polluting the torus

BlueGene/L Interconnection Networks



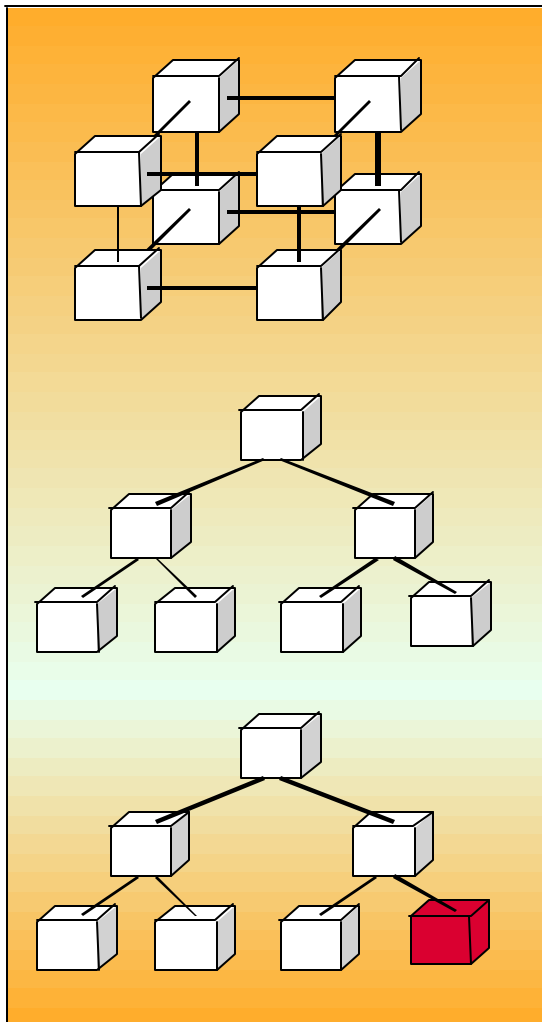
3 Dimensional Torus

- ❖ Interconnects all compute nodes (65,536)
- ❖ Virtual cut-through hardware routing
- ❖ 1.4Gb/s on all 12 node links (2.1 GB/s per node)
- ❖ Communications backbone for computations
- ❖ 350/700 GB/s bisection bandwidth

Global Tree

- ❖ One-to-all broadcast functionality
- ❖ Reduction operations functionality
- ❖ 2.8 Gb/s of bandwidth per link
- ❖ Latency of tree traversal in the order of 2 μ s
- ❖ Interconnects all compute and I/O nodes (1024)

BlueGene/L Interconnection Networks



3 Dimensional Torus

- ❖ Interconnects all compute nodes (65,536)
- ❖ Virtual cut-through hardware routing
- ❖ 1.4Gb/s on all 12 node links (2.1 GB/s per node)
- ❖ Communications backbone for computations
- ❖ 350/700 GB/s bisection bandwidth

Global Tree

- ❖ One-to-all broadcast functionality
- ❖ Reduction operations functionality
- ❖ 2.8 Gb/s of bandwidth per link
- ❖ Latency of tree traversal in the order of 2 μ s
- ❖ Interconnects all compute and I/O nodes (1024)

Ethernet

- ❖ Incorporated into every node ASIC
- ❖ Active in the I/O nodes (1:64)
- ❖ All external comm. (file I/O, control, user interaction, etc.)

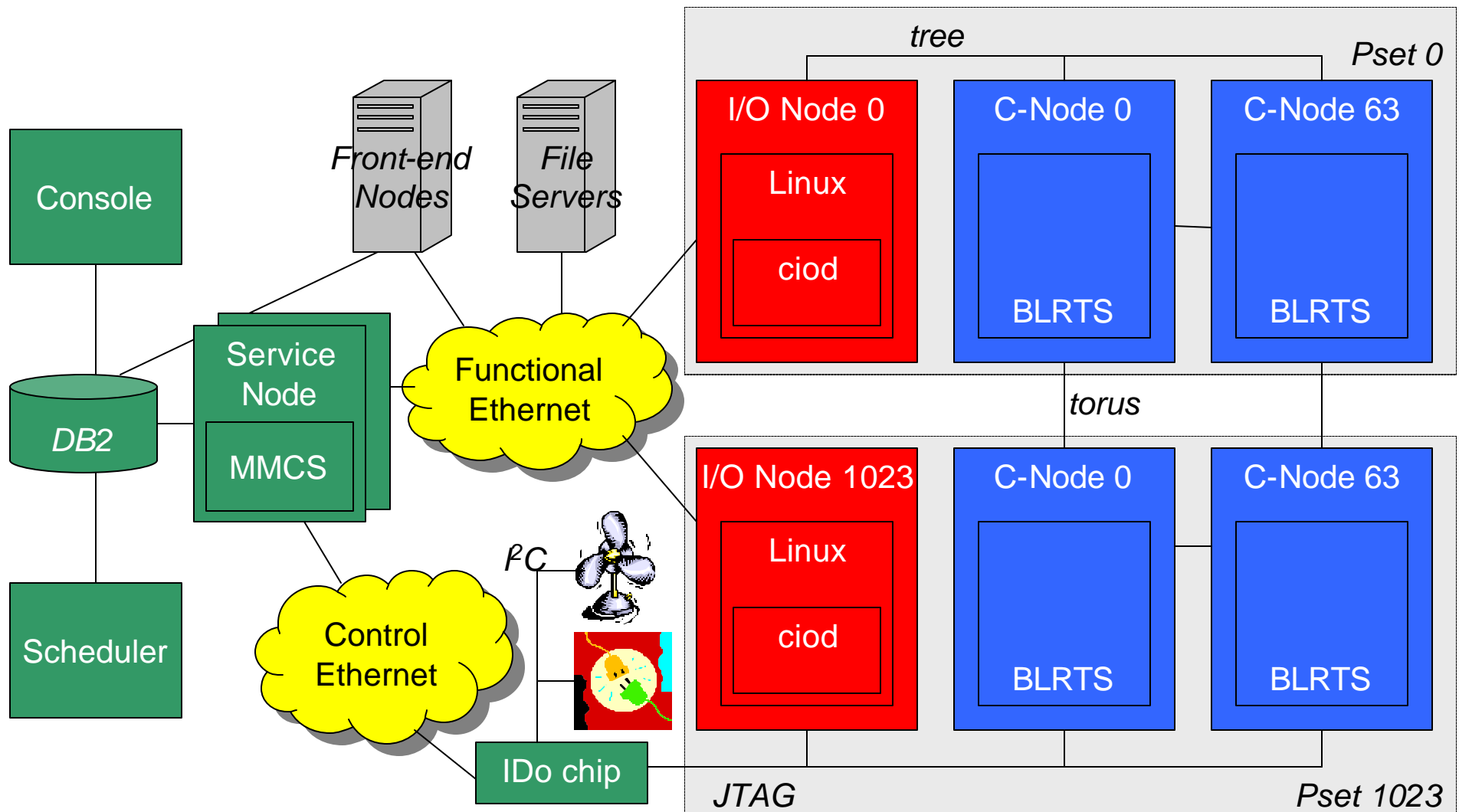
BlueGene/L Fundamentals (continued)

- Machine monitoring and control should also be offloaded
 - ❖ Machine boot, health monitoring
 - ❖ Performance monitoring
- Concept of a “service node”
 - ❖ Separate nodes dedicated to machine control and monitoring
 - ❖ Non-architected from a user perspective
- Control network
 - ❖ A separate network that does not interfere with application or I/O traffic
 - ❖ Secure network, since operations are critical
- By offloading services to other nodes, we let the compute nodes focus solely on application execution without interference
- BG/L is the extreme opposite of time sharing: everything is space shared

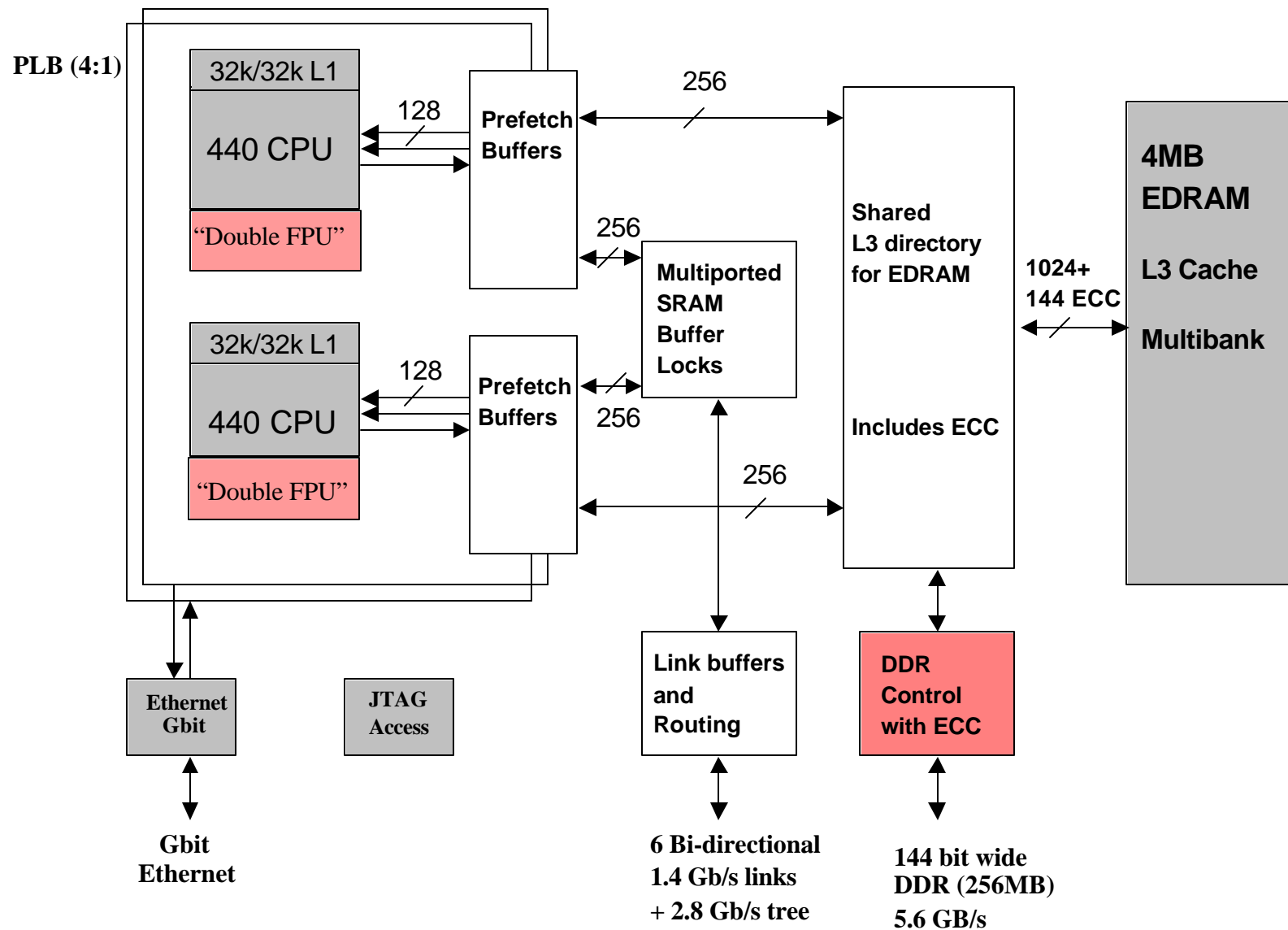
BlueGene/L Control System

- Control and monitoring are performed by a set of processes executing on the service nodes
- Non-architected network (Ethernet + JTAG) supports non-intrusive monitoring and control
- Core Monitoring and Control System
 - ❖ Controls machine initialization and configuration
 - ❖ Performs machine monitoring
 - ❖ Two-way communication with node operating system
 - ❖ Integration with database for system repository

Blue Gene/L System Architecture Overview



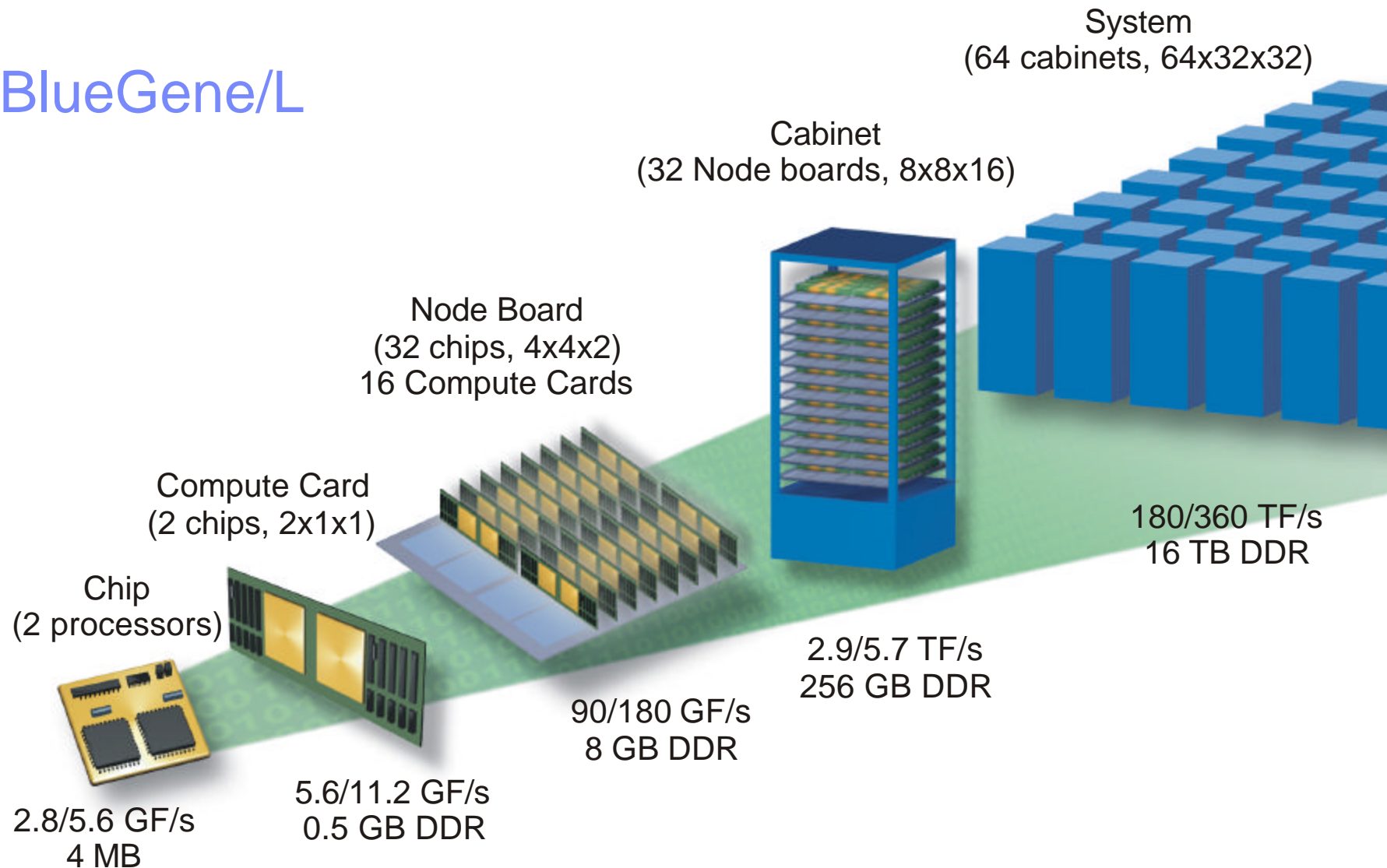
BlueGene/L Compute System-on-a-Chip ASIC



BlueGene/L Compute Chip

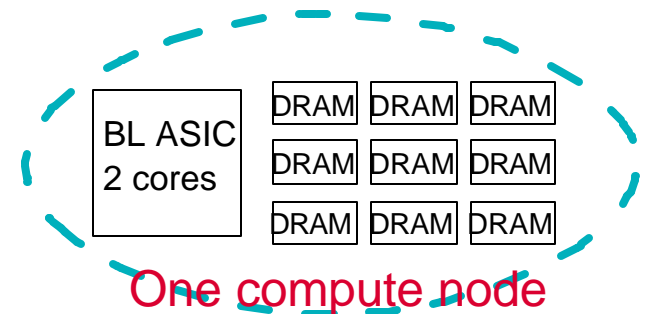
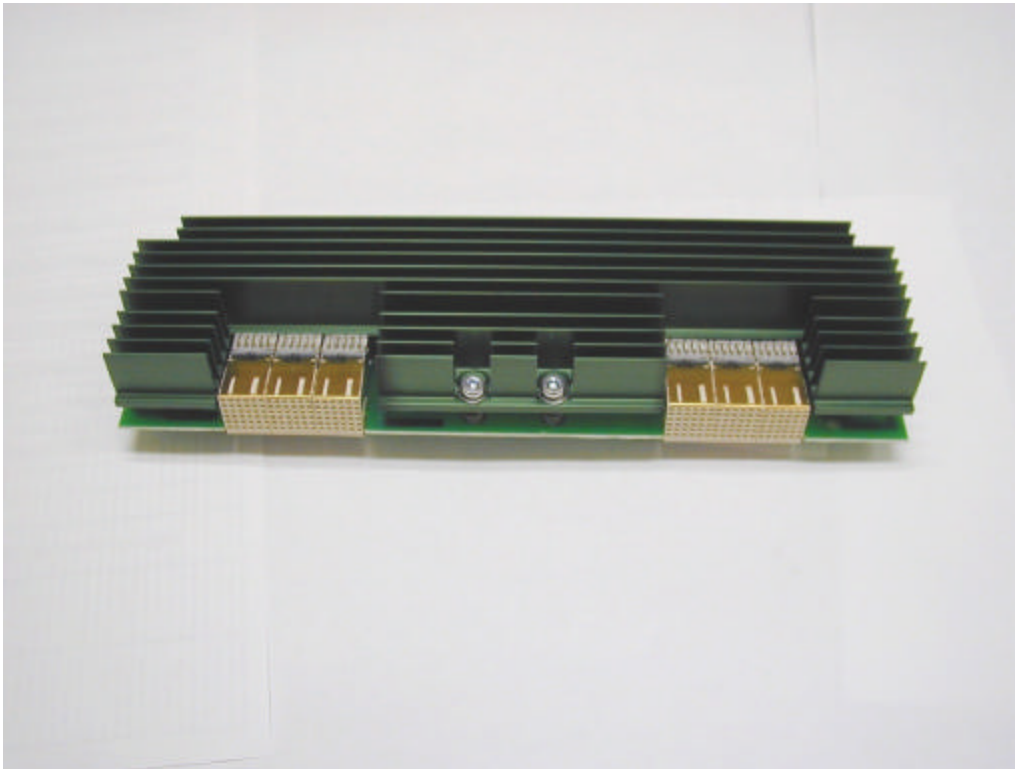
- Two PowerPC 440 32-bit processors @ 700 MHz
 - ❖ No L1 cache coherency between the processors
- New Hummer² FPU
 - ❖ operates on two-element vectors
 - ❖ 2 FMAs/FPU/cycle = 8 floating-point operations/cycle/chip
- Large on-chip shared cache (4 MB)
- High memory bandwidth/flop, communication/flop
 - ❖ 1 byte/flop of memory bandwidth
 - ❖ 0.375 byte/flop of communication bandwidth

BlueGene/L



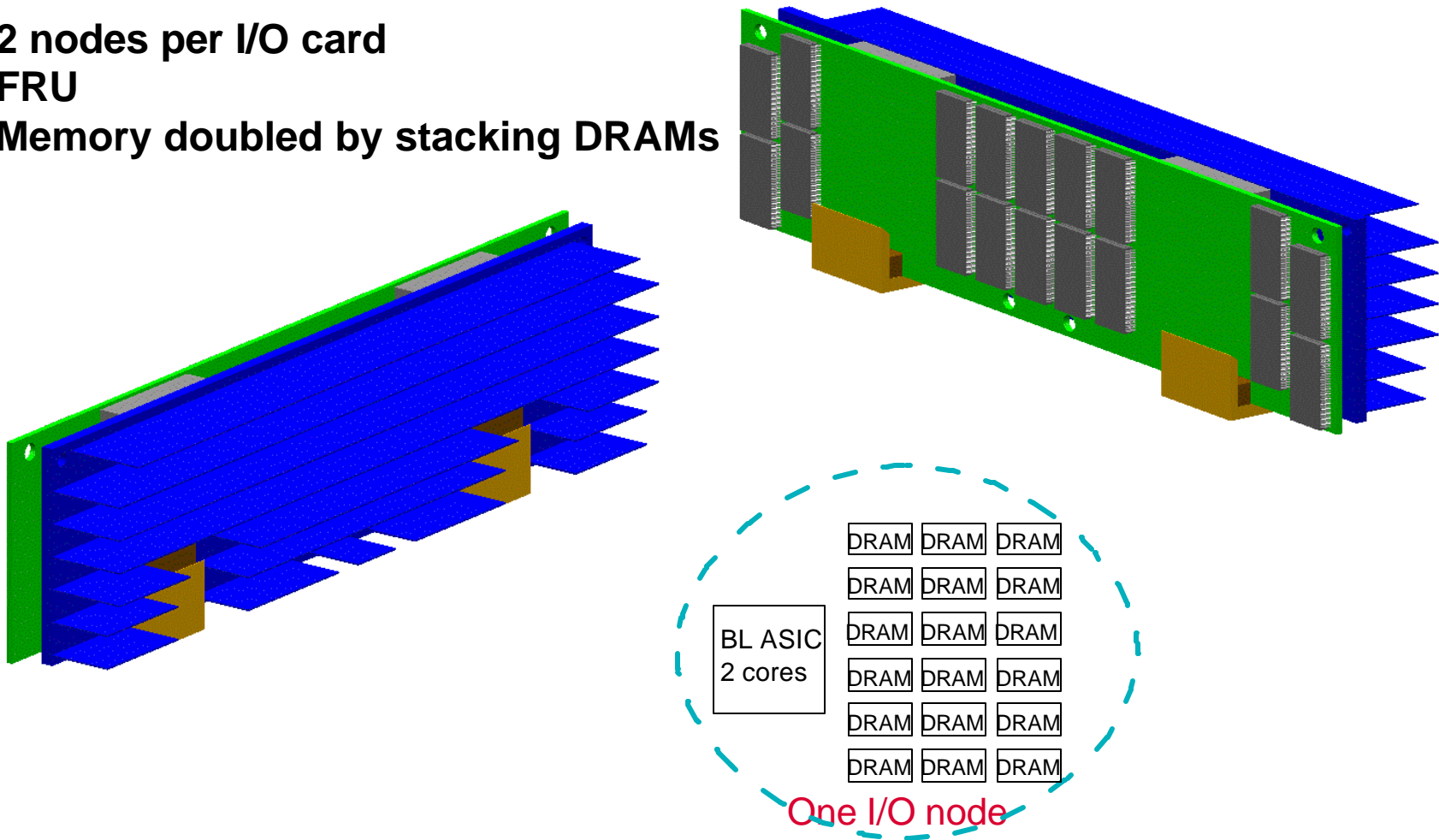
Compute Card

- 2 nodes per compute card
- Each BLC node in 25mm x 32mm CBGA
- FRU



I/O Card

- 2 nodes per I/O card
- FRU
- Memory doubled by stacking DRAMs



BlueGene/L

System
(64 cabinets, 64x32x32)

Cabinet
(32 Node boards, 8x8x16)

Node Board
(32 chips, 4x4x2)
16 Compute Cards

Compute Card
(2 chips, 2x1x1)

Chip
(2 processors)

2.8/5.6 GF/s
4 MB

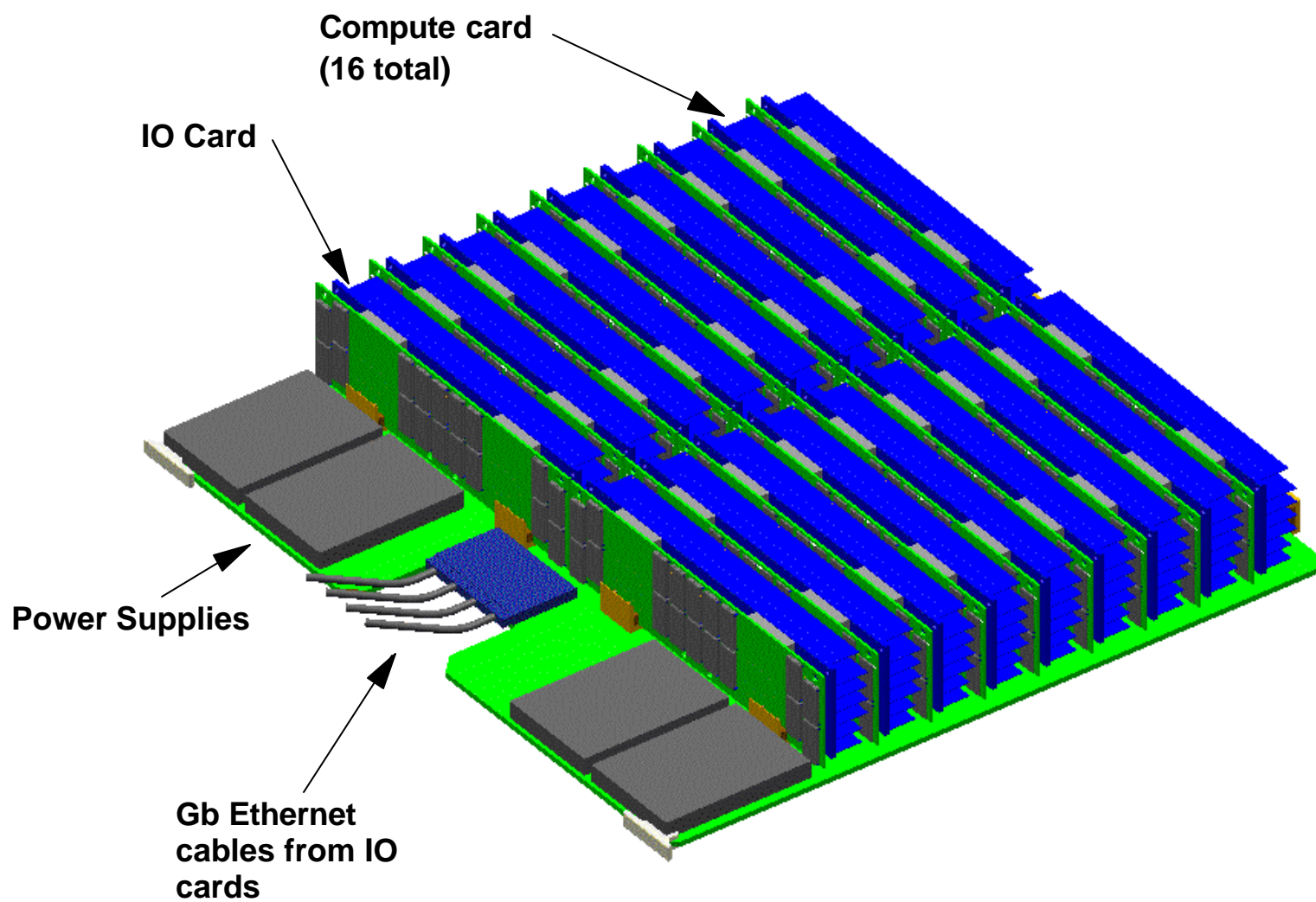
5.6/11.2 GF/s
0.5 GB DDR

90/180 GF/s
8 GB DDR

2.9/5.7 TF/s
256 GB DDR

180/360 TF/s
16 TB DDR

Node Card



BlueGene/L

System
(64 cabinets, 64x32x32)

Cabinet
(32 Node boards, 8x8x16)

Node Board
(32 chips, 4x4x2)
16 Compute Cards

Compute Card
(2 chips, 2x1x1)

Chip
(2 processors)

2.8/5.6 GF/s
4 MB

5.6/11.2 GF/s
0.5 GB DDR

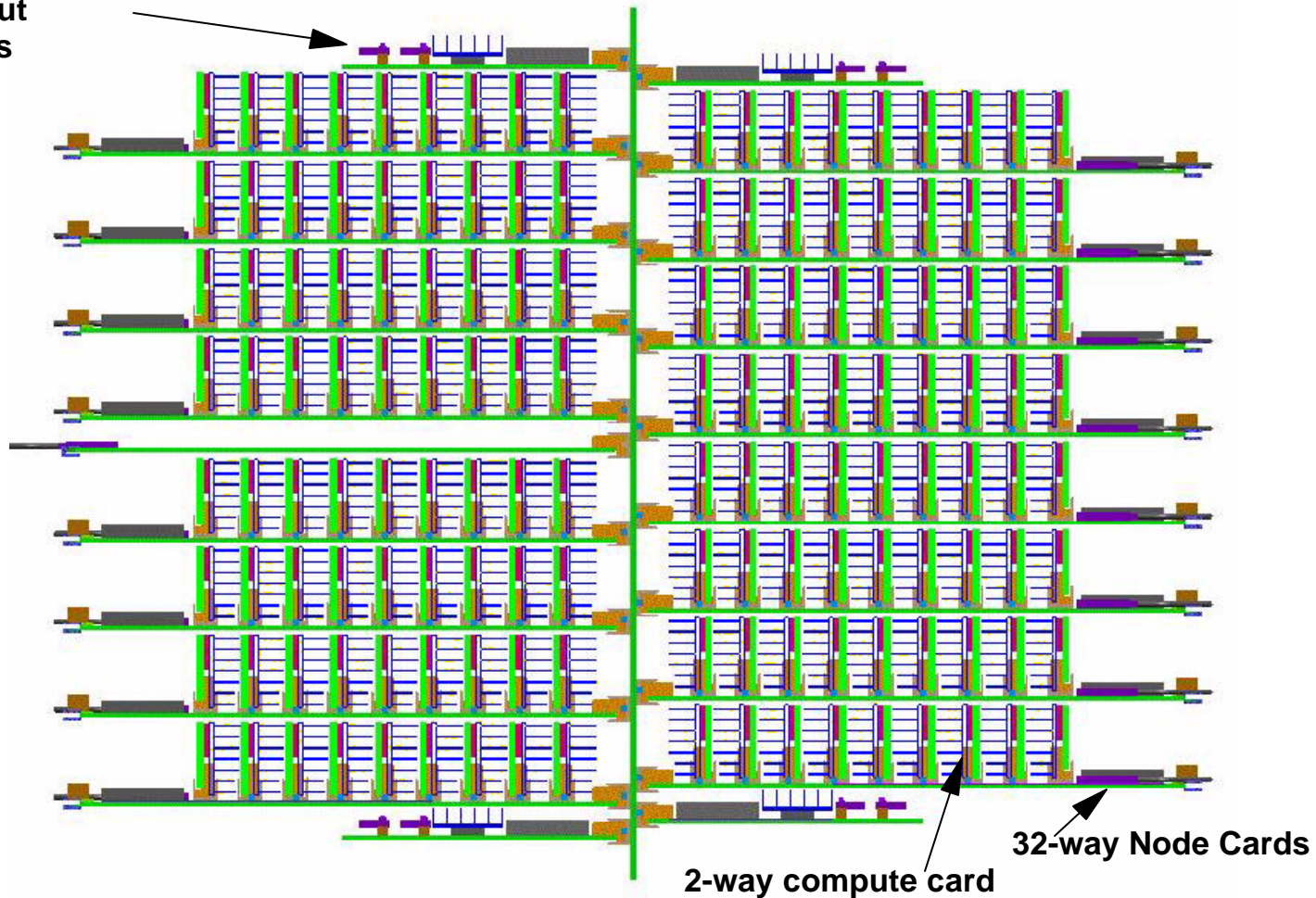
90/180 GF/s
8 GB DDR

2.9/5.7 TF/s
256 GB DDR

180/360 TF/s
16 TB DDR

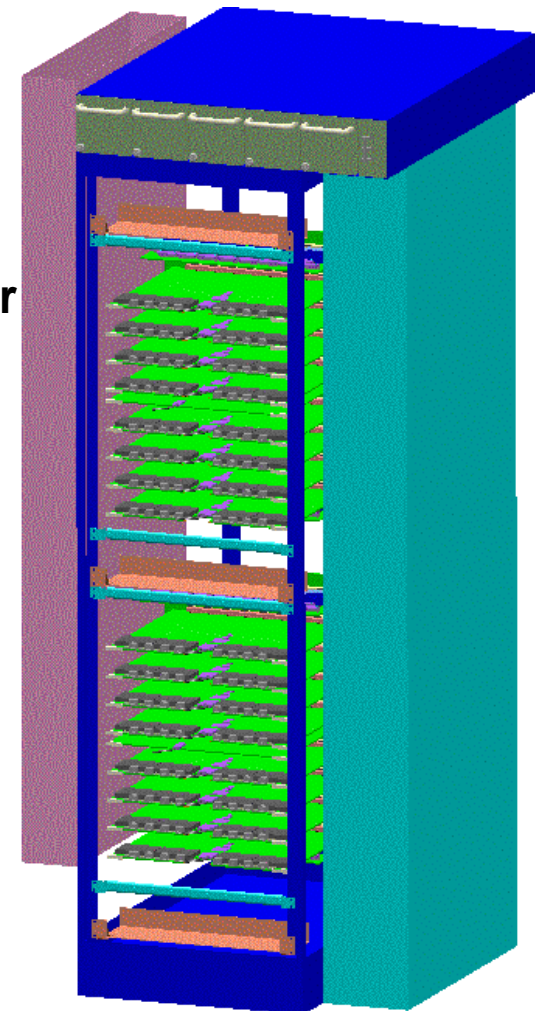
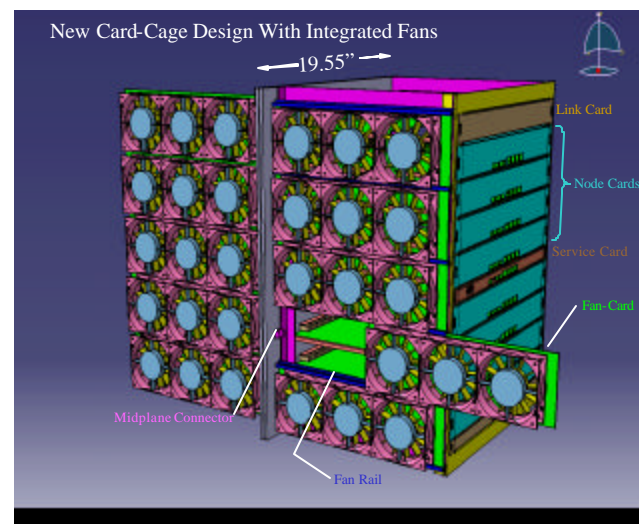
512-way Midplane, Side View

Link Cards
without
cables

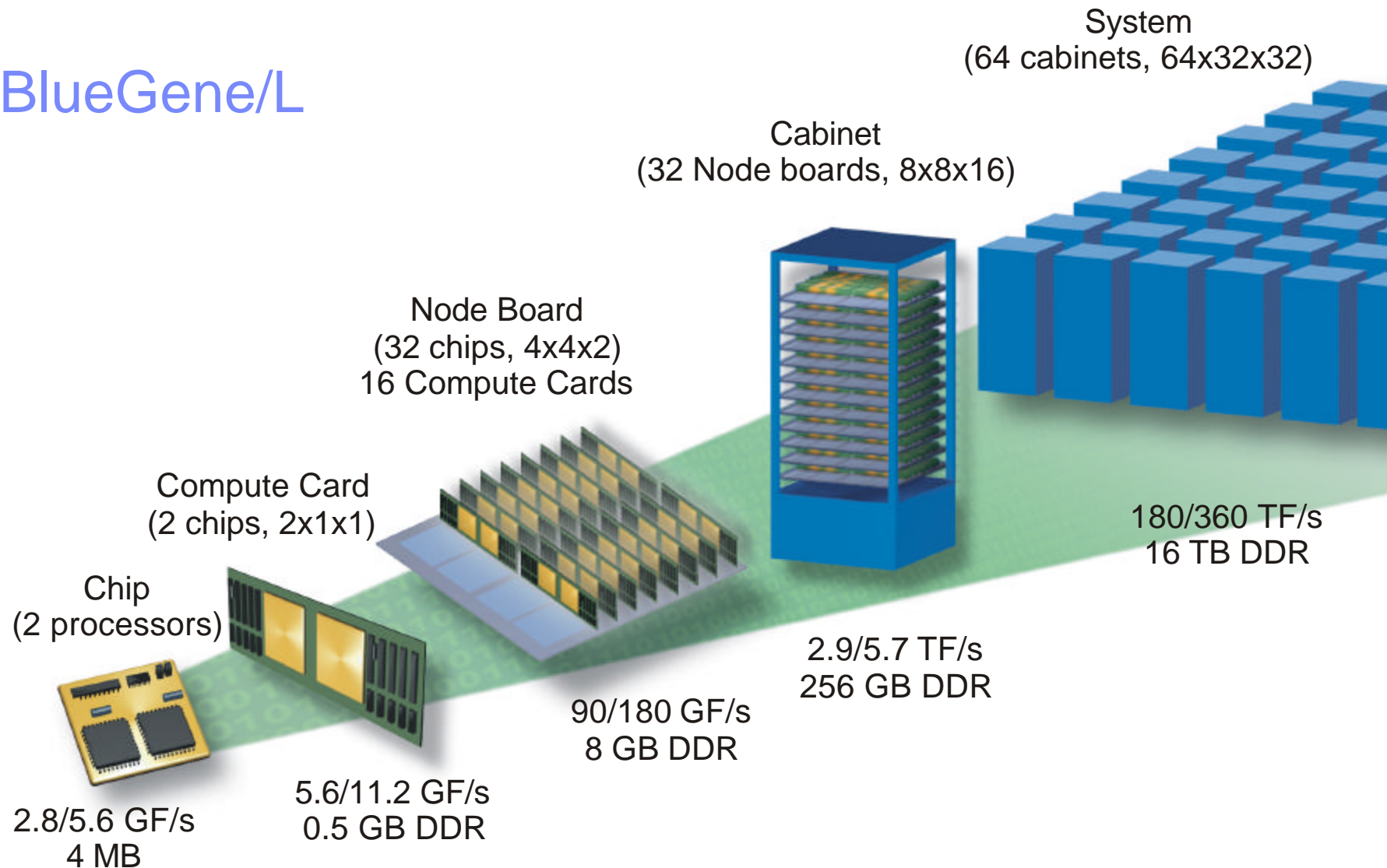


Rack, without Cables

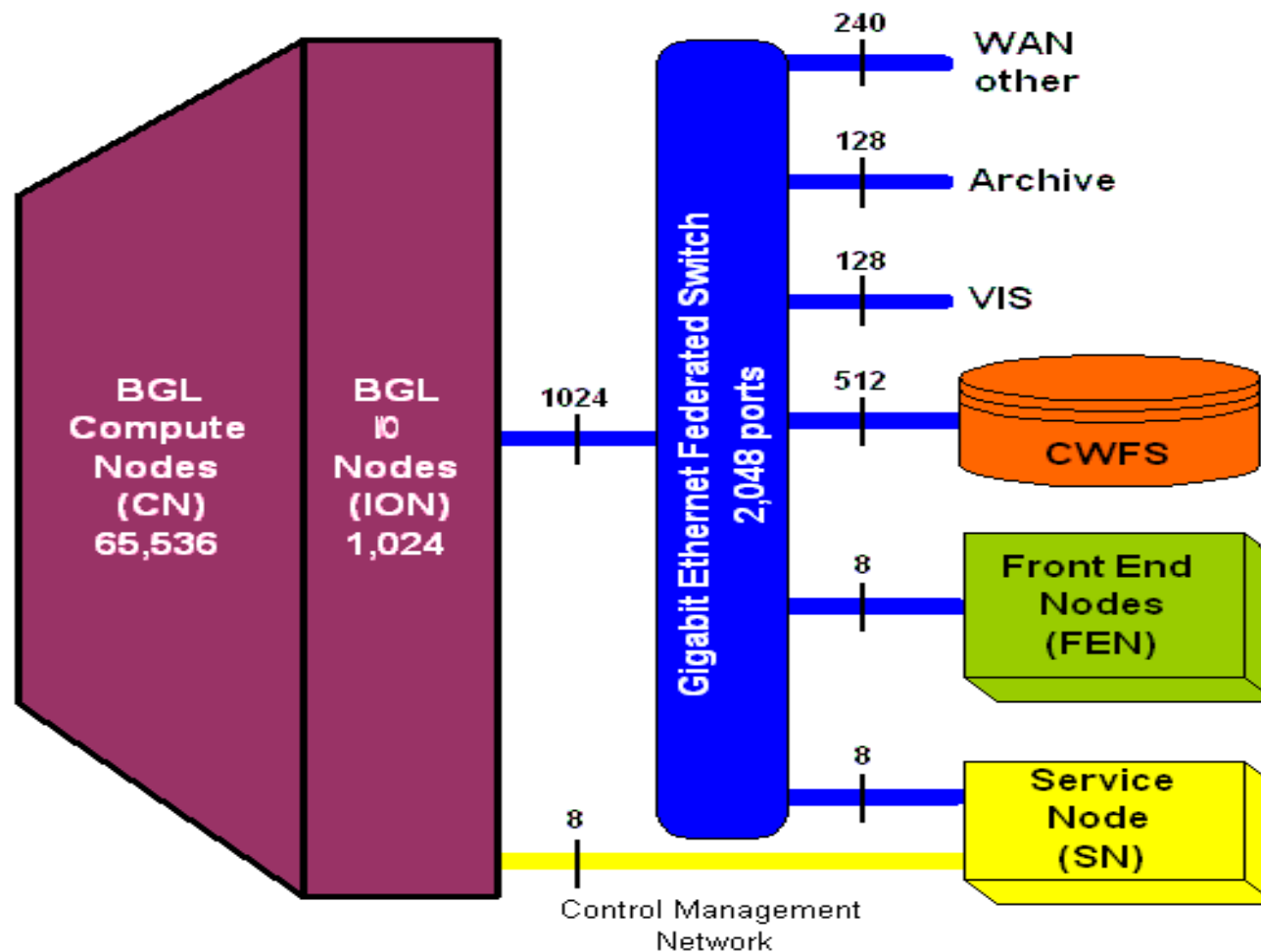
- 1024 compute processors
- 1024 communication processors
 - 256 GB DRAM
 - 2.8 TF peak
- 16 I/O nodes
 - 8 GB DRAM
 - 16 Gb Ethernet channels
- ~20 kW, air cooled
 - redundant power
 - redundant fans
- ~36"W x ~36"D x ~80" H (40-42U)



BlueGene/L

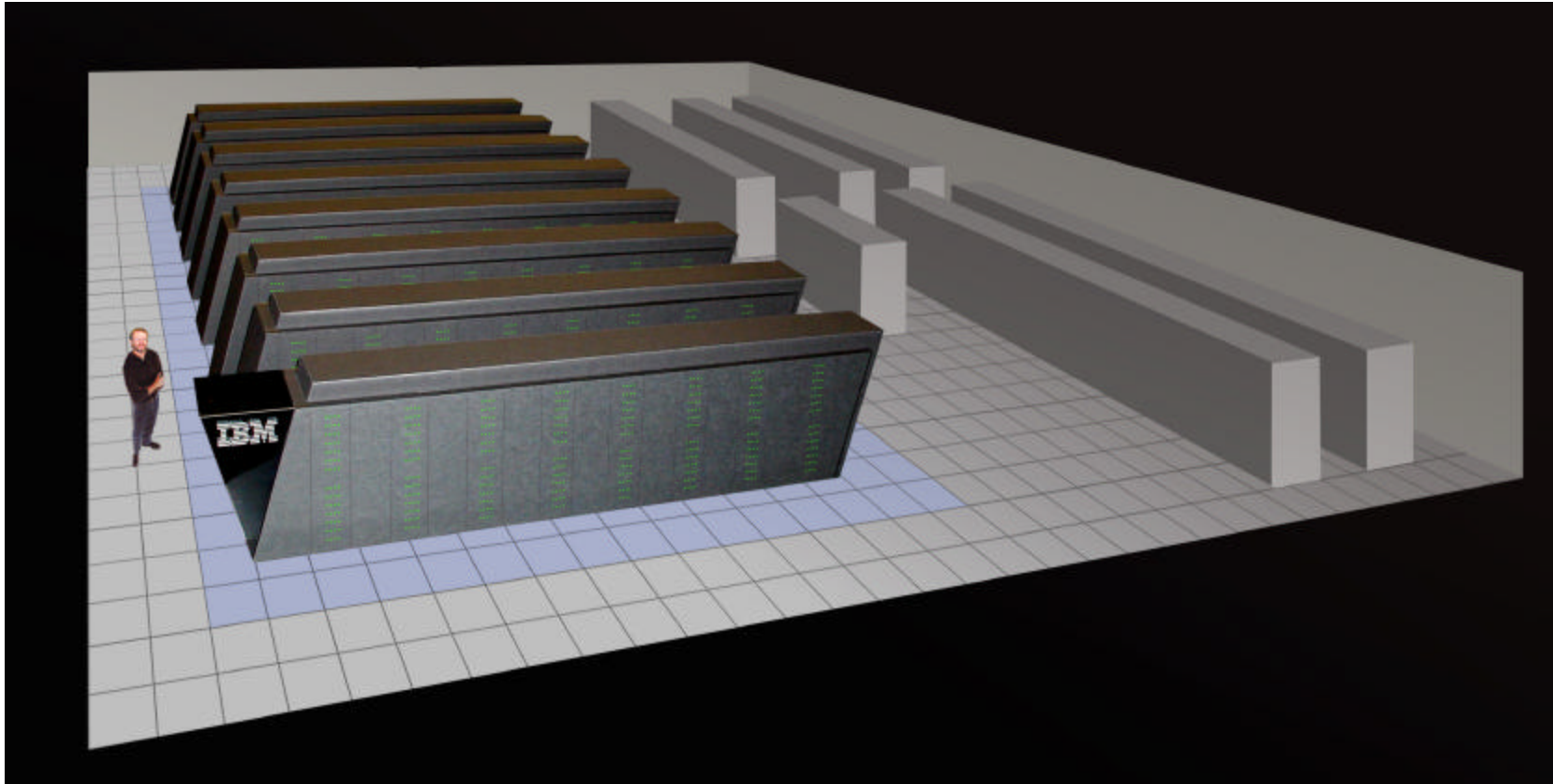


Complete BlueGene/L System at LLNL



Sep 16, 2002

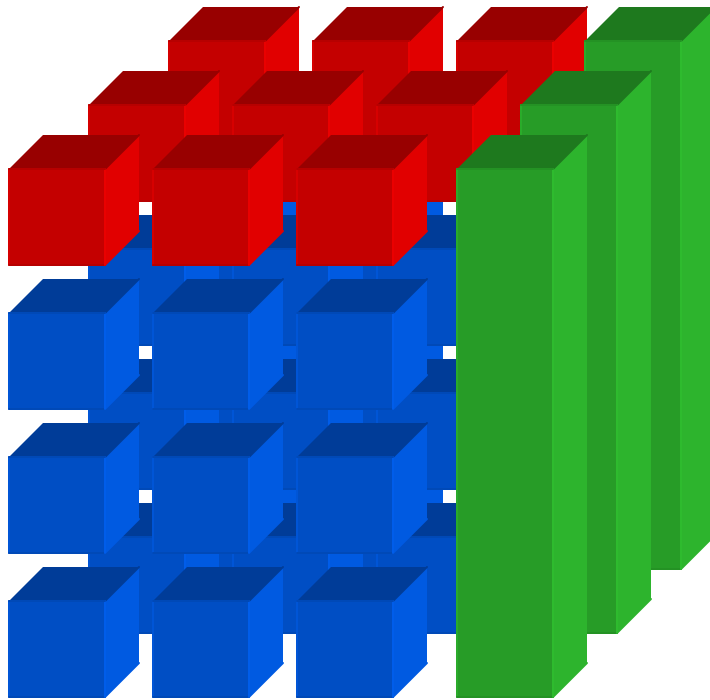
BlueGene/L System



Outline

- BlueGene/L high-level design philosophy
- BlueGene/L system architecture and technology overview
- **BlueGene/L software architecture**
- BlueGene/L in operation
- BlueGene/L programming

BlueGene/L Software Architecture



- User applications execute exclusively on the **compute nodes** and see only the **application volume** as exposed by the user-level APIs
- The outside world interacts only with the **I/O nodes** and processing sets (I/O node + compute nodes) they represent, through the **operational surface** – functionally, the machine behaves as a cluster of I/O nodes
- Internally, the machine is controlled through the **service nodes** in the **control surface** – goal is to hide this surface as much as possible

BlueGene/L Software Architecture Rationale

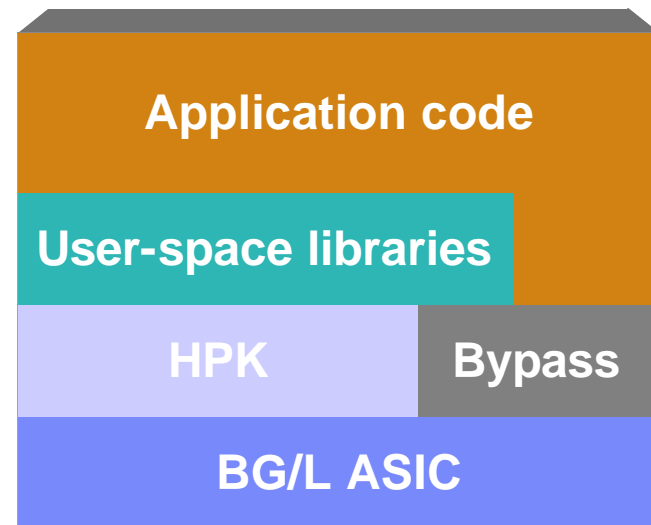
- We view the system as a cluster of 1,024 I/O nodes, thereby reducing a 65,536-node machine to a manageable size
- From a system perspective, user processes “live” in the I/O node:
 - ❖ Process management (start, monitor, kill)
 - ❖ Process debugging
 - ❖ Authentication, authorization
 - ❖ System management daemons (LoadLeveler, xCAT, MPI)
 - ❖ Traditional LINUX operating system
- User processes actually execute on compute nodes
 - ❖ Simple single-process (two threads) kernel on compute node
 - ❖ One processor/thread provides fast, predictable execution
 - ❖ User process extends into I/O node for complex operations
- Application model is a collection of private-memory processes communicating through messages

Programming Models for Compute Nodes

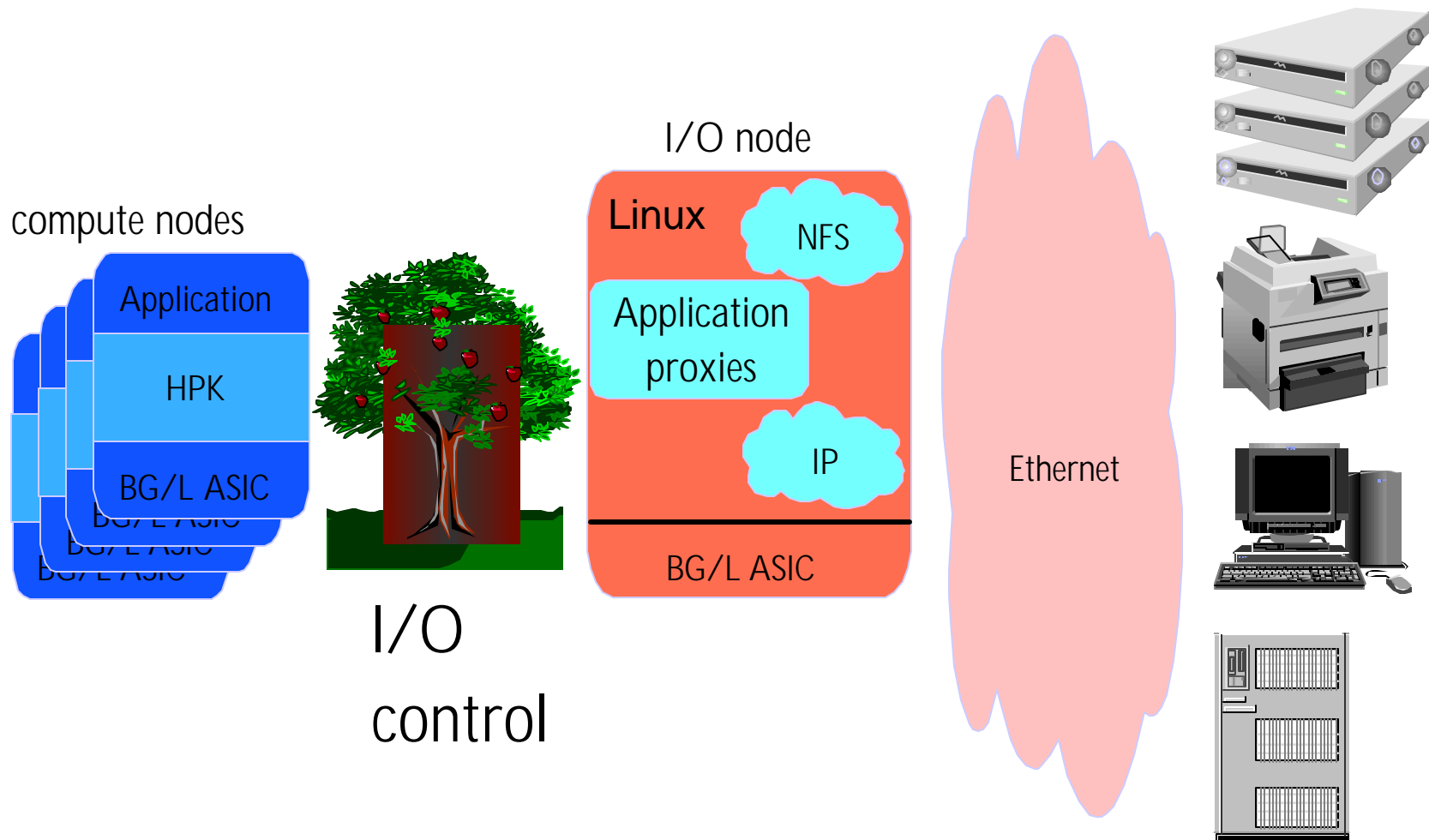
- **Heater mode:** CPU 0 does all the work for computation and communication, while CPU 1 spins idle (roles reversible)
 - ❖ Intended primarily for debugging mode of system
 - ❖ Some applications may actually benefit!
 - ❖ Reduces power consumption
- **Communication coprocessor mode:** CPU 0 executes user application while CPU 1 handles communications (roles reversible)
 - ❖ Preferred mode of operation for communication-intensive codes
 - ❖ Requires coordination between CPUs, which is handled in libraries
- **Symmetric mode:** Both CPUs do computation and communication within the same process
- **Virtual node mode:** Each CPU behaves as a separate node, executing the user application and also handling communication
 - ❖ attractive for compute-bound applications
 - ❖ separate processes (communicating via MPI) execute on CPU 0 and CPU 1
 - ❖ two single-threaded processes per compute node

Software Stack in BlueGene/L Compute Node

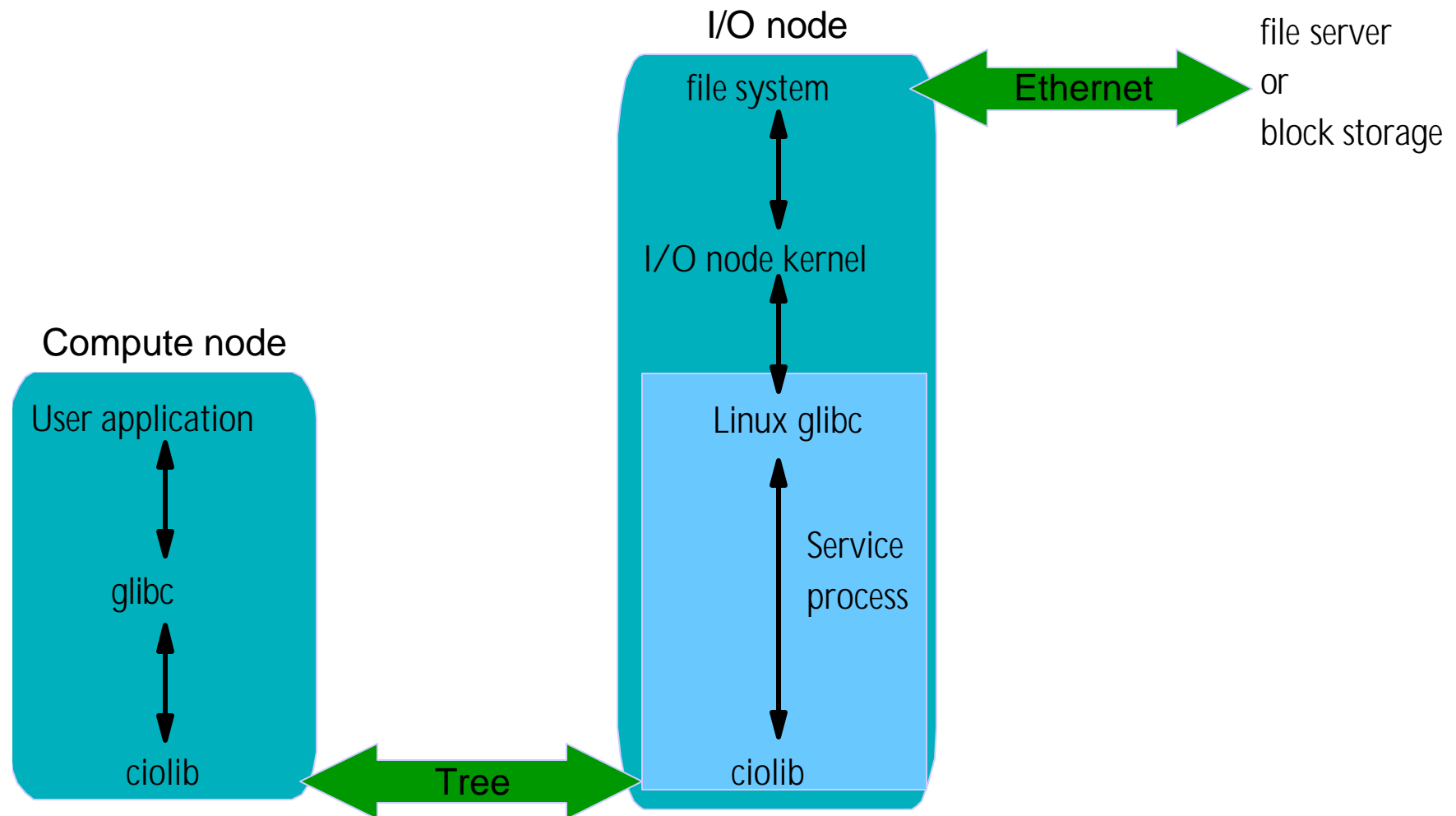
- HPK controls all access to hardware, and enables bypass for application use
- User-space libraries and applications can directly access torus and tree through bypass
- As a policy, user-space code should not directly touch hardware, but there is no enforcement of that policy
- Application code can use both processors in a compute node



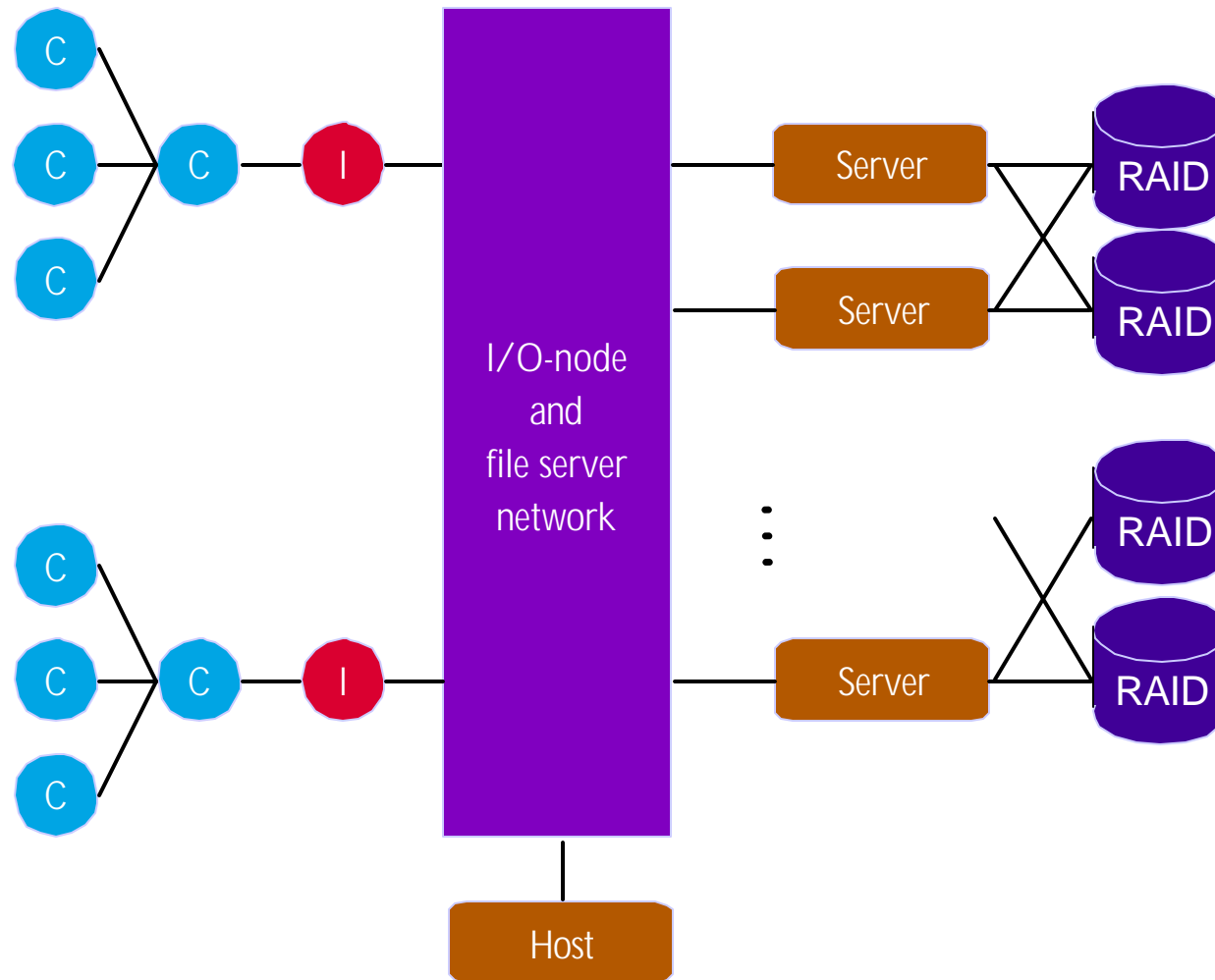
Role of I/O Nodes in System



File I/O in BlueGene/L



I/O Architecture: GPFS on a Cluster



Outline

- BlueGene/L high-level design philosophy
- BlueGene/L system architecture and technology overview
- BlueGene/L software architecture
- **BlueGene/L in operation**
- BlueGene/L programming

Communication Challenges

- Need to provide low-latency, high-bandwidth communication libraries to user application
 - ❖ Programming model is set of processes communicating through messages
- Software layers tend to add significant latency to system – matching sends and receives a major culprit
- Efficient implementation of MPI is a major goal

Communication Infrastructure for BlueGene/L

- Three layers of communication libraries
 - ❖ SPI active packets layer: directly maps to hardware
 - ❖ SPI active messages layer: abstraction layer
 - ❖ MPI: for end-user application
- Active packets and active messages are self-describing entities that cause actions to be executed on the receiving node
 - ❖ Packets have a maximum length determined by architecture (256 B)
 - ❖ Messages can be much longer and are broken down into packets
 - ❖ Can be used by applications, but are intended more for library developers
 - ❖ One-sided communication, which can be acted upon as soon as it is received by the target node
- MPI will be based on active messages layer
 - ❖ First step is a simple port of MPICH based on active messages
 - ❖ We will then enhance MPI with knowledge of machine topology
 - ❖ We will use the BG/L tree for global reductions and broadcasts

MPI on BlueGene/L (1)

- MPI is based on active messages layer, which is used as transport
- After a send/receive match, the sender can put the data directly in the destination space using an active message – good approach for large messages, as it avoids buffering and copies
- Buffers are still necessary for control and short messages

```
size = # of tasks in program
rank = my rank in set of tasks
if (rank == 0) {
    for (int i = 1; i < size; i++)
        receive from task i into buffer[i]
} else {
    send data to task 0
}
```

MPI on BlueGene/L (2)

- Typical solution requires dedicated buffers to each possible sender (all tasks) in every task and flow control
- 65,536 buffers of 1KB each would consume 25% of our memory, and not yield good performance
- A better solution is dynamic allocation of buffer space – only active senders get buffer space in receiver
- Monitor communication patterns and adapt
 - ❖ Reserve buffers based on recent history and inform sender about being prepared – avoid handshake even if “irecv” is not posted early
 - ❖ Give up reservation when communication patterns change
- Flow control is still needed, and packets can still be dropped!

MPI Issues

■ Alignment

- ❖ Packets pass through FPU – torus packet layer handles only aligned data
 - Use alignment pragmas, padding
- ❖ Extra copy needed if alignments differ at sender and receiver

■ Exploiting second processor as communication coprocessor

- ❖ Lack of coherence: Use coprocessor help at message layer level, for aligned “middle” packets

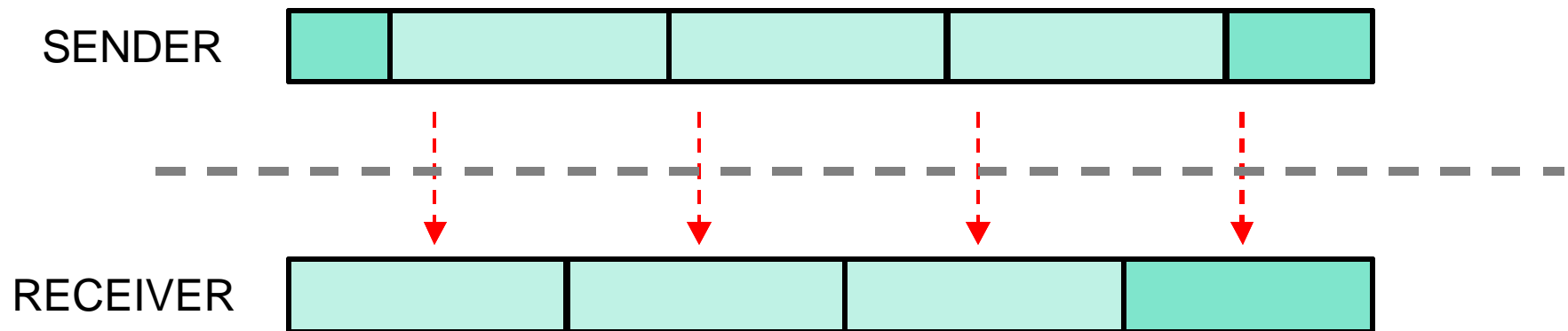
■ Scalable management of resources

- ❖ Dynamic pooling of open connections, buffers
- ❖ Adaptive management using history information

■ Efficient and scalable implementation of collective communications

- ❖ Use tree for global reduction/broadcast, torus multicast along a row
- ❖ Map 2D/3D Cartesian topology efficiently to torus

Alignment



- Problem: Torus packet layer only handles aligned data;
 - Message layer provides a data packetizer/unpacketizer
- When sender/receiver alignments are same, packet transmission is easy
 - head and tail transmitted in a single “unaligned” packet
 - aligned packets go directly to/from torus FIFOs
- When alignments differ, one extra memory copy is needed

Compilers

■ Support Fortran95, C99, C++

- ❖ GNU C, Fortran, C++ compilers can be used with BG/L, but they do not generate Hummer2 code
- ❖ IBM xlf/xlc compilers have been ported to BG/L, with code generation and optimization features for Hummer2 (licensing issues being resolved)

■ Backend enhanced to support PPC440 and to target SIMD FPU on nodes

- ❖ Finds parallel operations that match SIMD instructions
- ❖ Register allocator enhanced to handle register pairs
- ❖ Instruction scheduling tuned to unit latencies

■ Initial design of (2-way) SIMD FPU architecture was driven by key workload kernels such as matrix-matrix product and FFT

- ❖ Identified several mux combinations for SIMD operations not usually seen on other SIMD ISA extensions (Intel SSE, PPC AltiVec), e.g.,
$$d_P = a_P + b_P * c_P \parallel d_S = a_S - b_S * c_S$$
$$d_P = a_P + b_P * c_P \parallel d_S = a_S + b_P * c_S$$

Math Library

- Subset of ESSL (equivalent to LAPACK) being developed
 - ❖ Collaboration with Math dept and IRL
 - ❖ Mainly dense matrix kernels – DGEMM, DGEMV, DDOT, Cholesky and LU factorization
 - ❖ Early experience
 - Exposed problems with compiler – many known problems fixed
 - Concern about memory subsystem performance
- Status
 - ❖ DGEMM (matrix multiplication) getting excellent performance, estimated at ~ 2.4 GFLOPS on a 700 MHz node (86% of peak; VHDL simulations with hot L1 cache show: 93% of peak on both CPUs; With cold L1 cache: 76% of peak)

Event Analysis and Failure Prediction

■ Failures in large clusters

- ❖ Most of the failures are addressed through manual intervention as they are noticed
- ❖ Most of the actions taken are based on individual experiences and decisions

■ Goals for BG/L: Use event prediction techniques to predict and prevent critical failures, either automatically or with minimal human intervention

■ Basic approach

- ❖ Monitor event (failures) logs from the machine
- ❖ Develop models from observation (control theory)
- ❖ Predict future events (failures)
- ❖ Use failure prediction to schedule maintenance, guide process migration

Fault Recovery Based on Checkpointing

- Different levels of transparency to user application
 - ❖ User controlled: Application does self checkpointing
 - ❖ Semi-automatic: Application indicates program points for checkpointing, checkpoint performed by system software
 - ❖ Automatic: Completely transparent checkpointing by system software
- Application launched on an error-free partition after loading data from previous checkpoint
- Approach insufficient for dealing with undetected soft errors – may occur once a month for 65,536-node machine
 - ❖ Consistency checks within application
 - ❖ Error checking by system software – in general, need redundancy

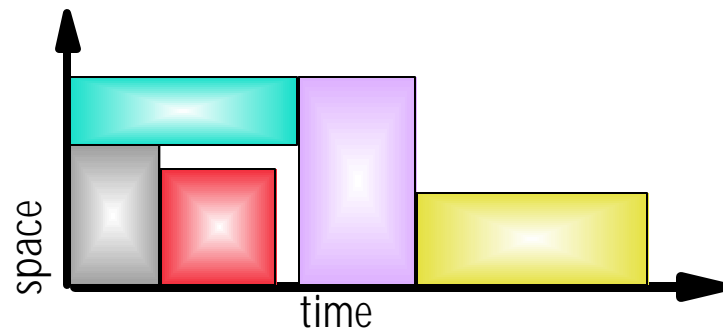
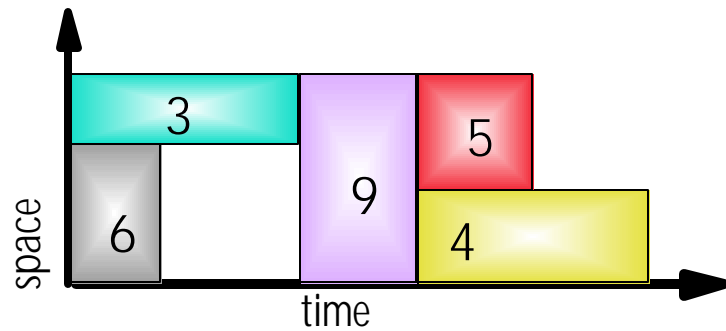
Job Scheduling in BlueGene/L

- Job scheduling strategies can significantly impact the utilization of large computer systems
- Many large systems see utilization in the 50-70% range
- Machines with toroidal topology (as opposed to all-to-all switch) are particularly sensitive to job scheduling – this was demonstrated at LLNL with gang scheduling on Cray T3D
- For BlueGene/L, we have investigated the impact of two scheduling techniques
 - ❖ Task migration
 - ❖ Backfilling

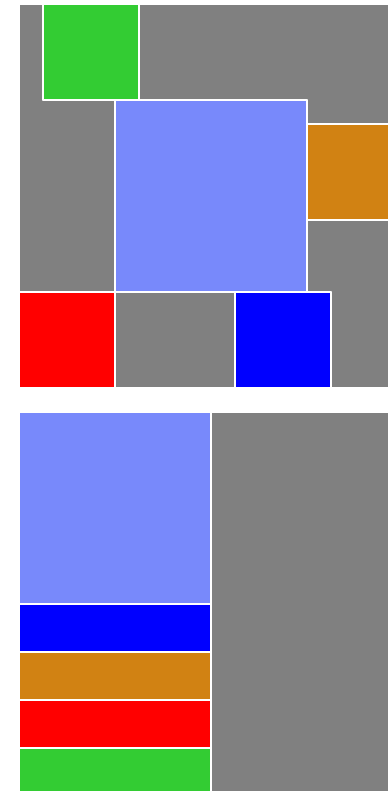
Task Migration and Backfilling

backfilling

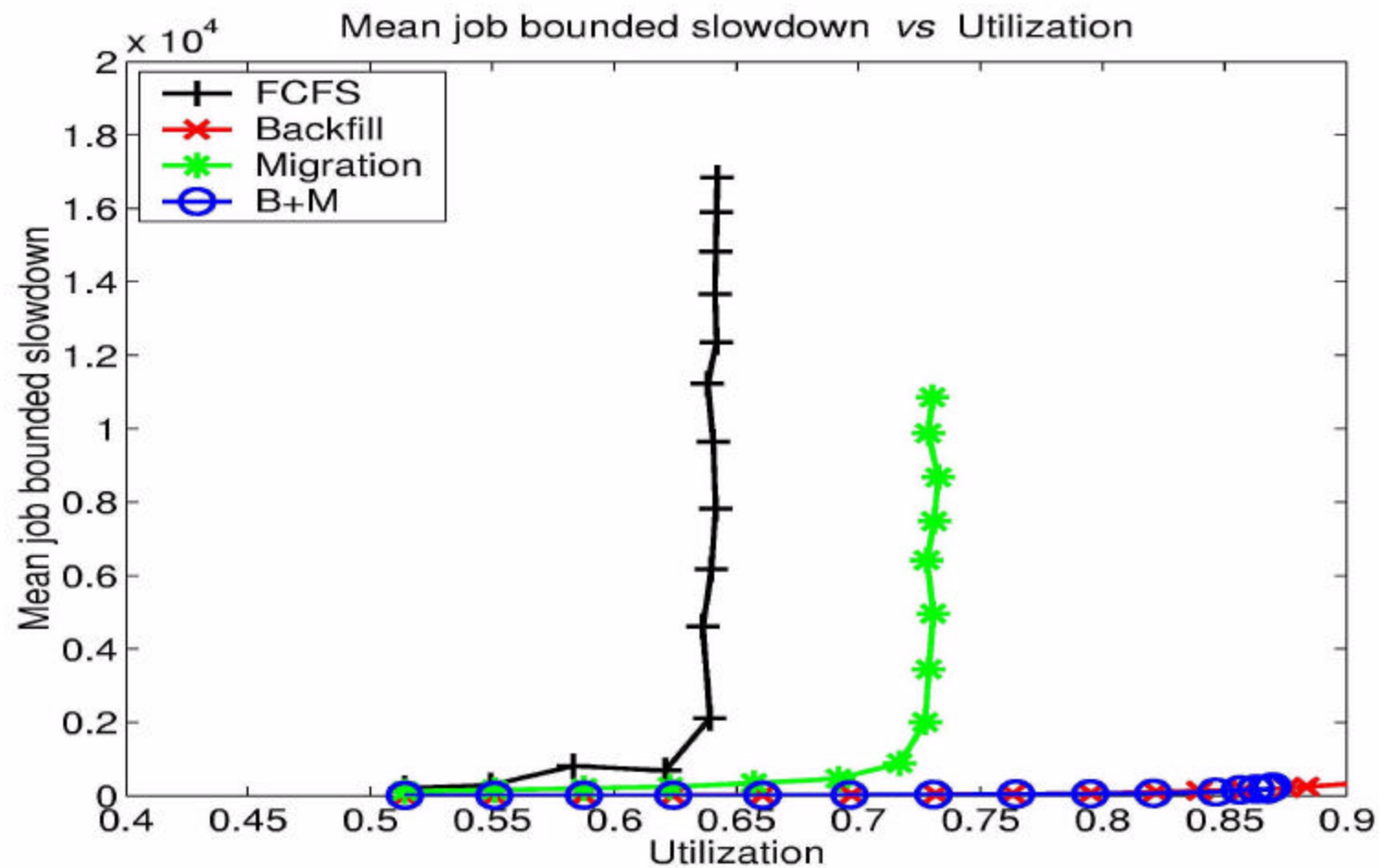
queue = {6,3,9,5,4}, 10 nodes



migration



Results for Job Scheduling on BlueGene/L



BGLsim: System-Level Simulator

- Complete system-level simulator for BlueGene/L
 - ❖ Based on the Mambo simulation infrastructure from Austin Research
 - ❖ Complements our VHDL simulators (which are much slower)
- Architecturally accurate simulator
 - ❖ Executes the full BG/L instruction set, including Hummer²
 - ❖ Models all of the devices, including Ethernet, torus, tree, lock box, etc.
 - ❖ Supports development of system software and applications
 - ❖ Currently not performance aware: counts only instructions, not cycles
 - ❖ We have simulated systems with up to 64 nodes (not a limitation)
- Efficient simulation that supports code development
 - ❖ 1,000,000 – 2,000,000 BG/L instructions per second on 1GHz Pentium III
 - ❖ Can be extended to 5-10% performance accurate without much drop
- We have used it to develop/test HPK, Linux, device drivers, networking, MPI, compilers, FPU benchmarks

Outline

- BlueGene/L high-level design philosophy
- BlueGene/L system architecture and technology overview
- BlueGene/L software architecture
- BlueGene/L in operation
- **BlueGene/L programming**



IBM Research

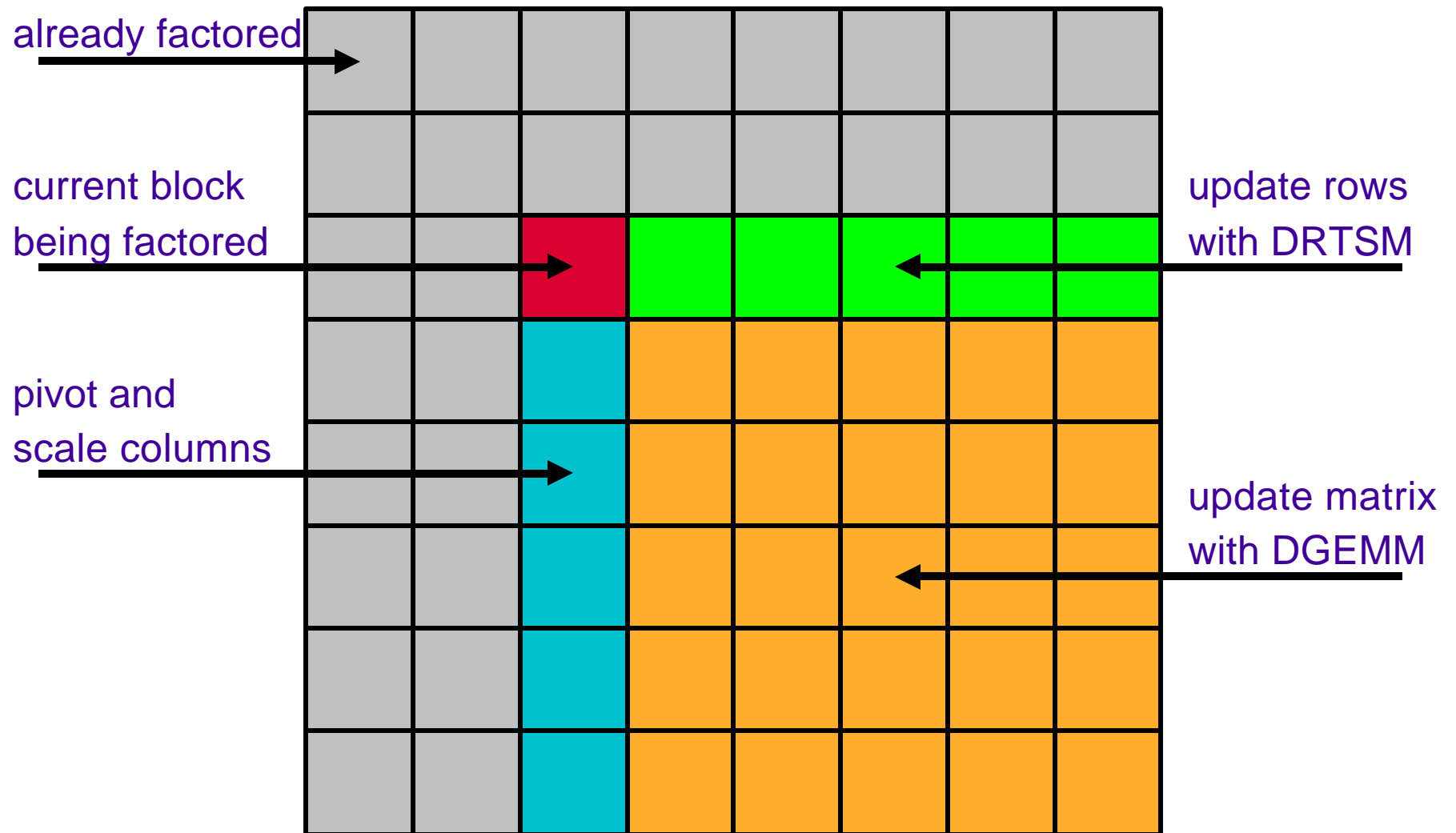
Case study: Parallel LU factorization on BG/L

or
how I stopped worrying and learned to love BG/L

Making Good Use of BG/L with MPI

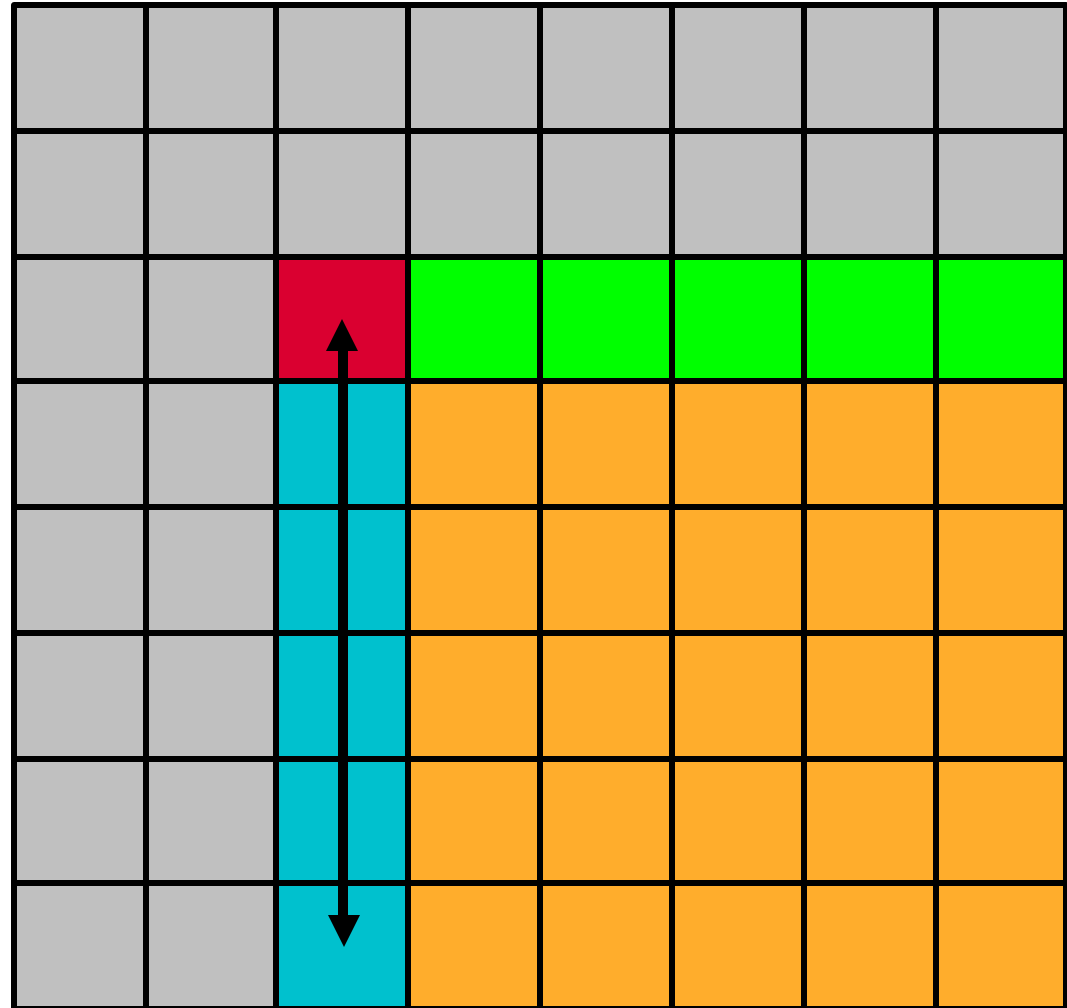
- IBM's BlueGene/L is a feature-rich architecture:
 - ❖ Torus interconnect for high-bandwidth communication
 - ❖ Multicast operations along any axis of the torus
 - ❖ Tree interconnect for low-latency/high-bandwidth reduces/broadcasts
 - ❖ 16-byte floating-point unit capable of 4 floating-point operations per cycle
 - ❖ Two processors/node
- Exploiting those features requires several levels of support
 - ❖ Compilers must be able to extract and expose instruction-level parallelism
 - ❖ Libraries must be coded so that they use the features
 - ❖ Application programs must be coded so that features can be used
- This case study is about organizing your MPI program so that it can take advantage of the "cool" features of BlueGene/L
- It is organized as a case study of a well-know computation: Blocked LU factorization

Blocked LU Factorization – An Algorithm



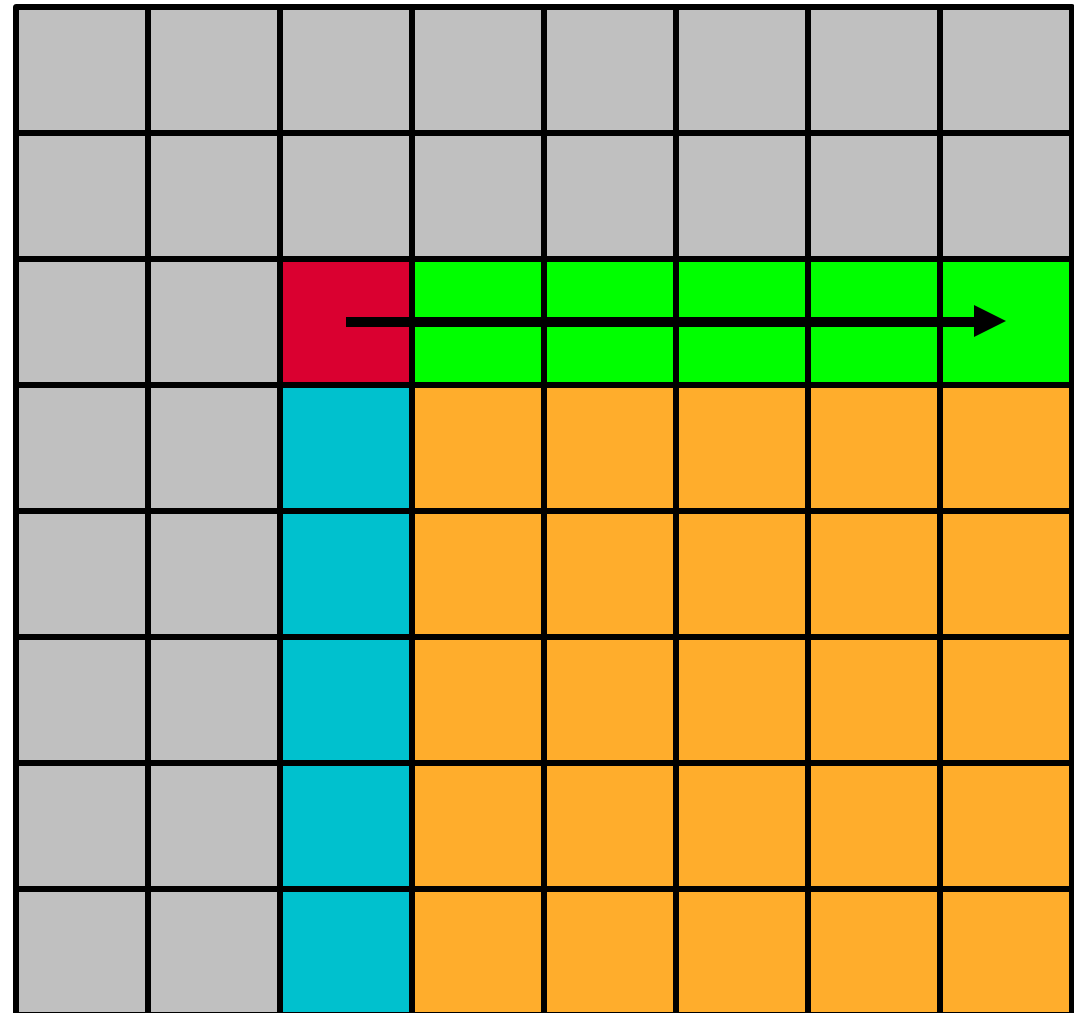
Blocked LU Factorization – Step 1

- Start with next diagonal block
- Pivot and update columns
 - ❖ find maximum absolute value
 - ❖ exchange with diagonal
 - ❖ update next columns
- At the end, **red block** is in LU
- factored form, **blue blocks** are updated



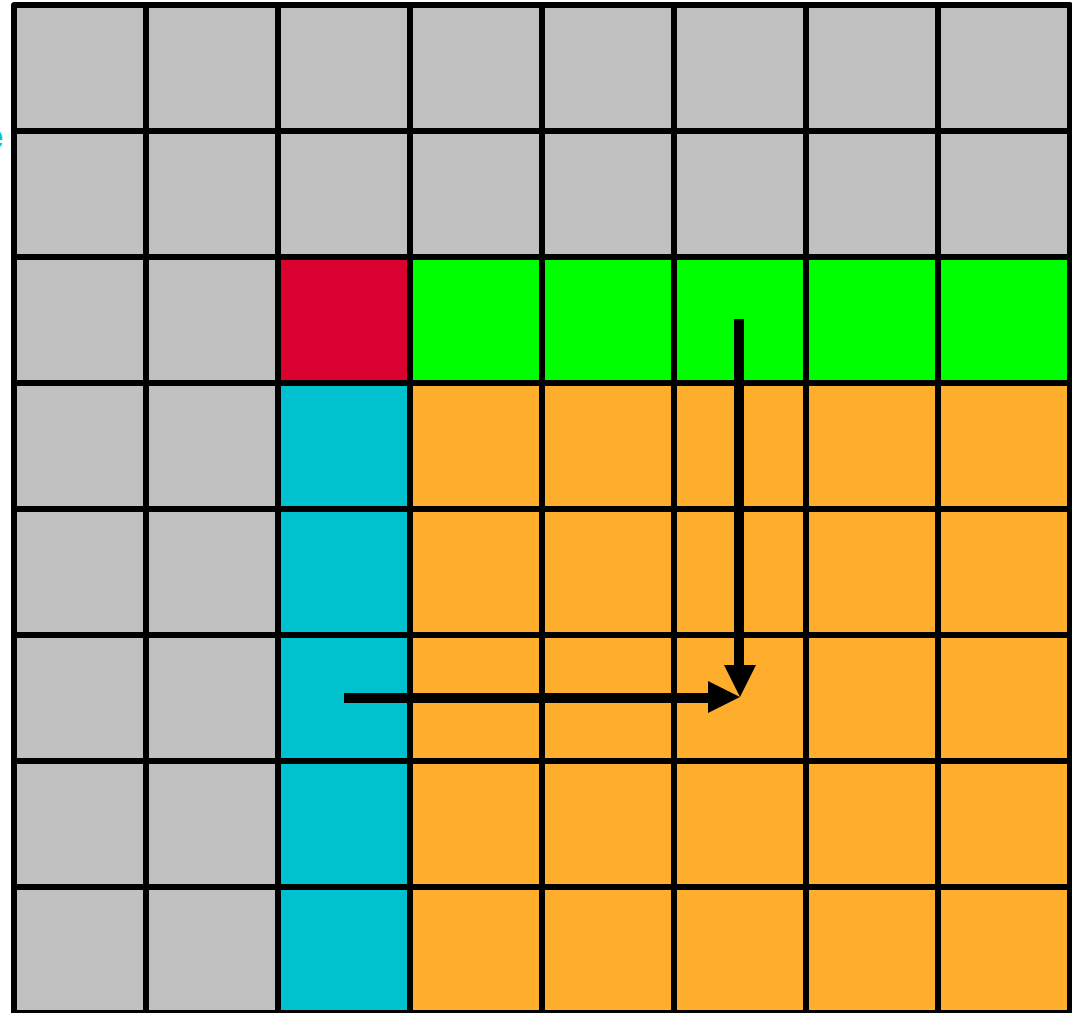
Blocked LU Factorization – Step 2

- Update row of green blocks
 - ❖ use lower factor of red block with each green block
 - ❖ DTRSM operation
- At the end, green blocks are updated



Blocked LU Factorization – Step 3

- Update each orange block
 - ❖ use green block of column and blue block of row
 - ❖ DGEMM operation
- Each blue block is used in every orange block of row
- Each green block is used in every orange block of column
- At the end, orange blocks are updated
- LU factorization continues with next diagonal block



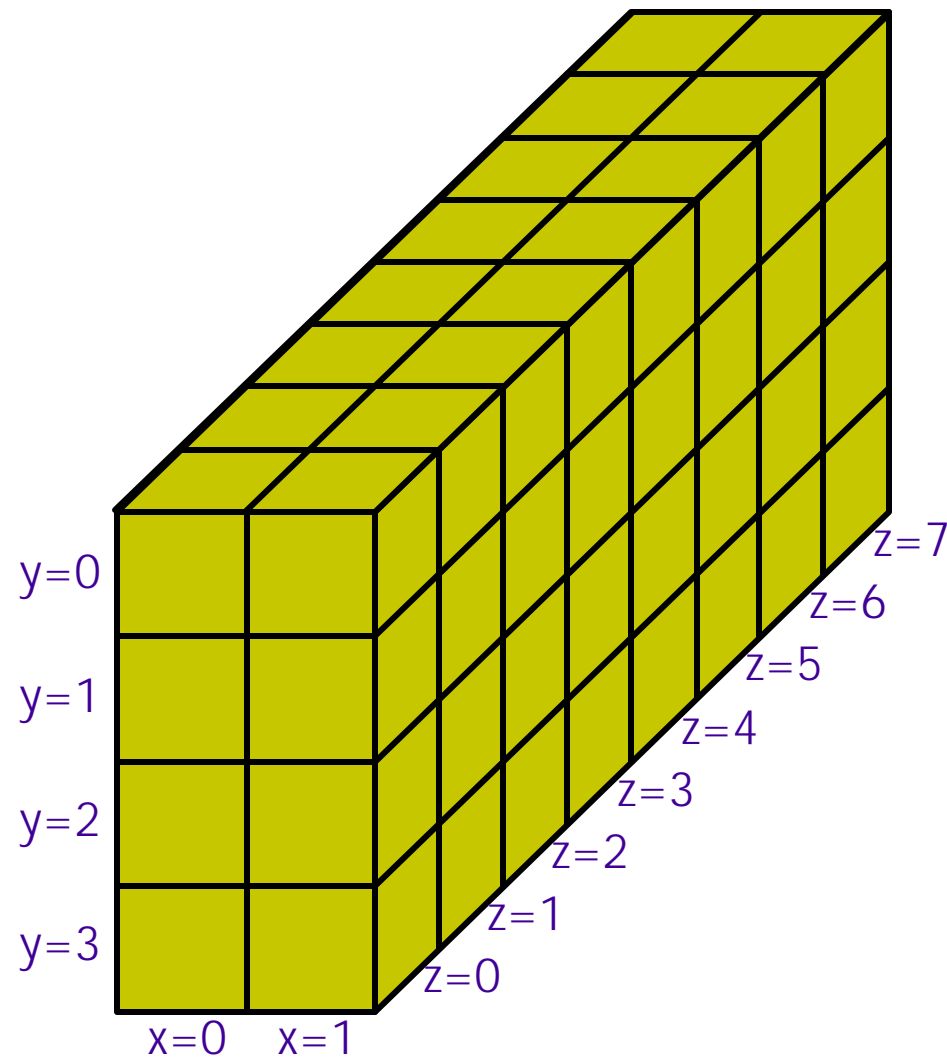
Parallel LU Factorization

- We partition the matrix as an 8x8 array of blocks
- We execute the factorization on an 8x8 logical grid of processes
- The data is decomposed using a block distribution
- (In practice, block-cyclic distribution gives better utilization)
- To run on BG/L:
 - ❖ request the appropriate number of compute nodes
 - ❖ create a logical two-dimensional grid of processes
 - ❖ perform operations on that logical grid

$P_{0,0}$	$P_{1,0}$	$P_{2,0}$	$P_{3,0}$	$P_{4,0}$	$P_{5,0}$	$P_{6,0}$	$P_{7,0}$
$P_{0,1}$	$P_{1,1}$	$P_{2,1}$	$P_{3,1}$	$P_{4,1}$	$P_{5,1}$	$P_{6,1}$	$P_{7,1}$
$P_{0,2}$	$P_{1,2}$	$P_{2,2}$	$P_{3,2}$	$P_{4,2}$	$P_{5,2}$	$P_{6,2}$	$P_{7,2}$
$P_{0,3}$	$P_{1,3}$	$P_{2,3}$	$P_{3,3}$	$P_{4,3}$	$P_{5,3}$	$P_{6,3}$	$P_{7,3}$
$P_{0,4}$	$P_{1,4}$	$P_{2,4}$	$P_{3,4}$	$P_{4,4}$	$P_{5,4}$	$P_{6,4}$	$P_{7,4}$
$P_{0,5}$	$P_{1,5}$	$P_{2,5}$	$P_{3,5}$	$P_{4,5}$	$P_{5,5}$	$P_{6,5}$	$P_{7,5}$
$P_{0,6}$	$P_{1,6}$	$P_{2,6}$	$P_{3,6}$	$P_{4,6}$	$P_{5,6}$	$P_{6,6}$	$P_{7,6}$
$P_{0,7}$	$P_{1,7}$	$P_{2,7}$	$P_{3,7}$	$P_{4,7}$	$P_{5,7}$	$P_{6,7}$	$P_{7,7}$

Creating a Physical Node Partition

- Node partitions are created when jobs are scheduled for execution
- User specifies desired shape when submitting job:
 - ❖ **submit lufact 2x4x8**
 - ❖ request a job partition of 64 compute nodes, with shape 2 (on x-axis) by 4 (on y-axis) by 8 (on z-axis)
- A contiguous, rectangular subsection of the compute nodes is carved out for this job
- Nodes are indexed by their (x,y,z) coordinates inside the job partition



Mapping Processes to Physical Nodes

- In MPI, logical process grids are created with **MPI_CART_CREATE**
- The mapping is performed by the system, matching physical topology
- In this case, we have mapped each xy-plane to one column
- Within a column, consecutive values of y are neighbors
- Logical row operations correspond to operations on a string of physical nodes along the z-axis
- Logical column operations correspond to operations on an xy-plane
- row and column communicators are created with **MPI_CART_SUB**

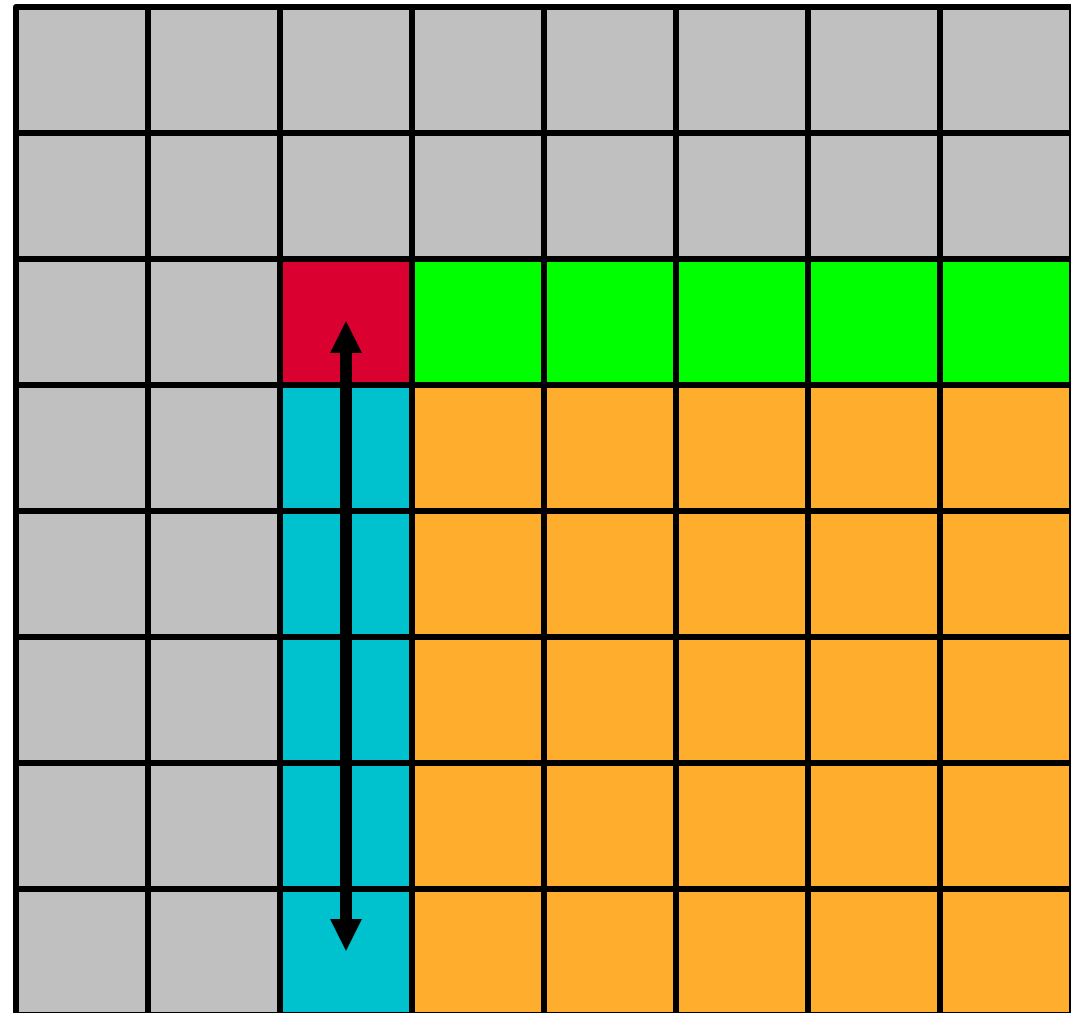
0,0,0	0,0,1	0,0,2	0,0,3	0,0,4	0,0,5	0,0,6	0,0,7
0,1,0	0,1,1	0,1,2	0,1,3	0,1,4	0,1,5	0,1,6	0,1,7
0,2,0	0,2,1	0,2,2	0,2,3	0,2,4	0,2,5	0,2,6	0,2,7
0,3,0	0,3,1	0,3,2	0,3,3	0,3,4	0,3,5	0,3,6	0,3,7
1,0,0	1,0,1	1,0,2	1,0,3	1,0,4	1,0,5	1,0,6	1,0,7
1,1,0	1,1,1	1,1,2	1,1,3	1,1,4	1,1,5	1,1,6	1,1,7
1,2,0	1,2,1	1,2,2	1,2,3	1,2,4	1,2,5	1,2,6	1,2,7
1,3,0	1,3,1	1,3,2	1,3,3	1,3,4	1,3,5	1,3,6	1,3,7

Creating Communicators

- First, create a two-dimensional 8x8 cartesian communicator:
 - ❖ **GRID2D_COMM = MPI_CART_CREATE(MPI_COMM_WORLD, 2, 8x8)**
- Then, create a communicator along the row of each process:
 - ❖ **ROW_COMM = MPI_CART_SUB(GRID2D_COMM, [true, false])**
- Finally, create a communicator along the column of each process:
 - ❖ **COL_COMM = MPI_CART_SUB(GRID2D_COMM, [false, true])**

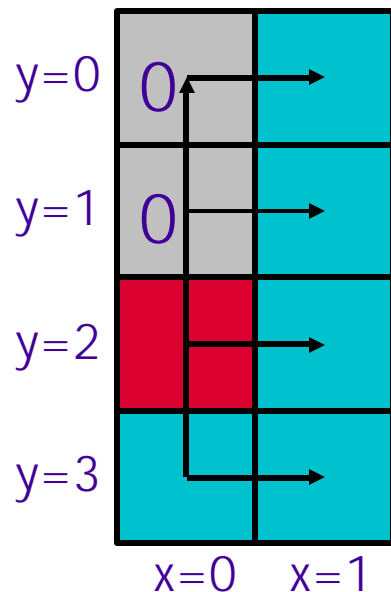
Blocked LU Factorization – Step 1

- Pivot and update columns
 - ❖ find maximum absolute value
 - ❖ exchange with diagonal
 - ❖ update next columns
- At the end, **red block** is in LU factored form, **blue blocks** are updated
- Each process finds its maximum absolute value in column
- **MPI_REDUCE** is used to find column maximum
 - ❖ operation on a logical column of processes = xy-plane in
 - ❖ physical partition



Computing Global Maximum

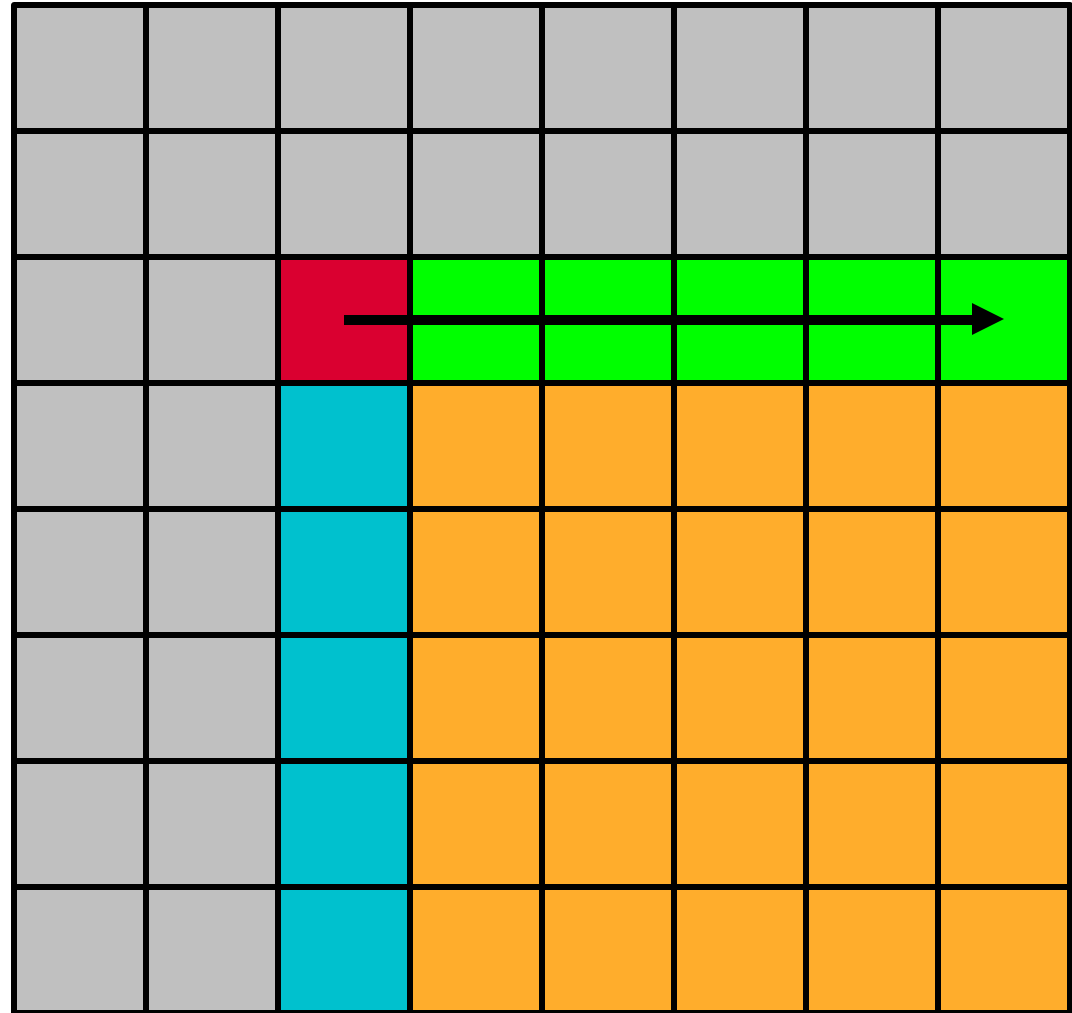
- Expressing operation in MPI:
pivot = MPI_REDUCE(local_max, MAX, COL_COMM)
- Performing the operation on a physical plane:



- One option is to map the reduction to the tree:
 - ❖ use different classes for subsets of processes
 - ❖ in general, cannot be done for all desired subsets
- Another option is to use row multicast to flood the plane with local maxima
 - ❖ takes advantages of high bandwidth in torus
- Or, can just use point-to-point communications to perform reduction as data is moved

Blocked LU Factorization – Step 2

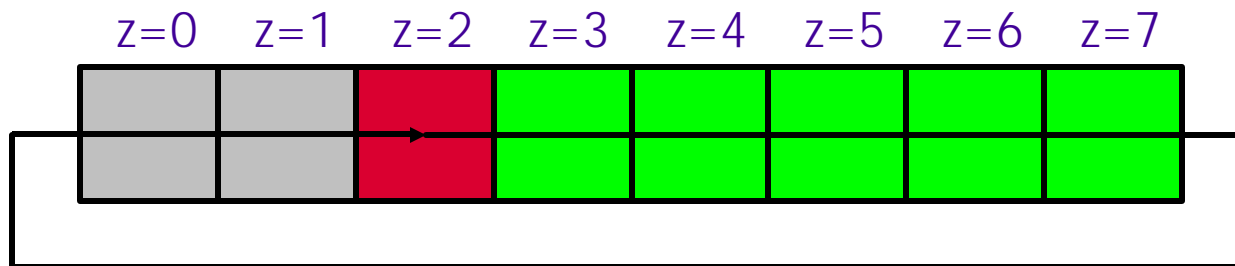
- Update row of **green blocks**
 - ❖ use lower factor of **red block** with each **green block**
 - ❖ DTRSM operation
- At the end, **green blocks** are updated
- **MPI_BCAST** is used to distribute factor:
 - ❖ operation on logical row of processes = z-axis of physical partition



Broadcasting Along Logical Row

- Expressing operation in MPI:
MPI_BCAST(block, ROW_COMM)

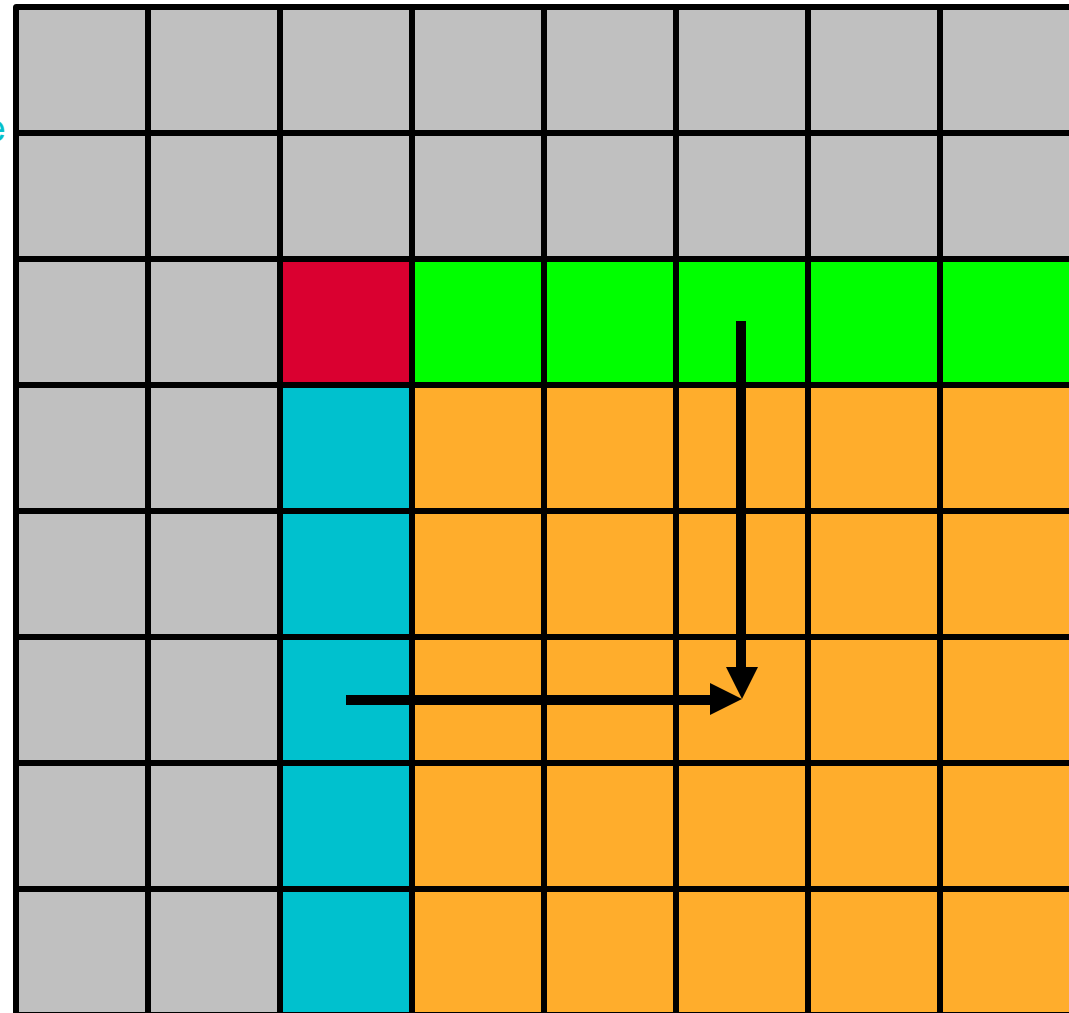
- Performing operation on a physical string of nodes:



- A simple multicast to all nodes in z-axis will do the trick

Blocked LU Factorization – Step 3

- Update each **orange block**
 - ❖ use **green block** of column and **blue block** of row
 - ❖ DGEMM operation
- Each **blue block** is used in every **orange block** of row
- Each **green block** is used in every **orange block** of column
- At the end, **orange blocks** are updated
- LU factorization continues with next diagonal block
- **MPI_BCAST** is used to distribute **blue blocks** along row and **green blocks** along columns



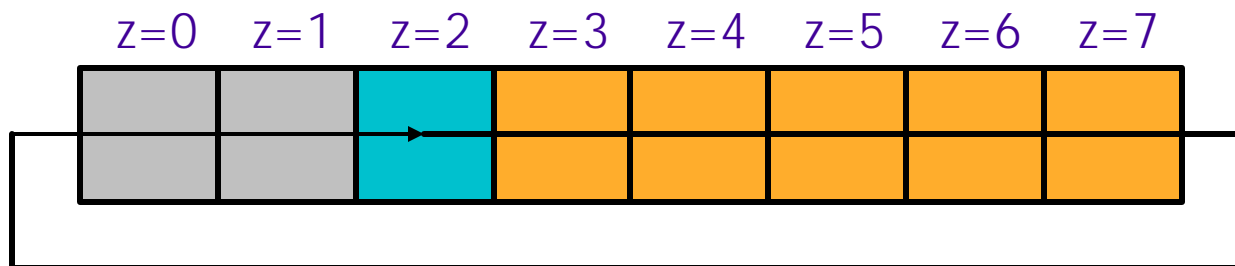
Broadcasting Along Logical Rows/Columns

- Expressing operation in MPI:

MPI_BCAST(**block**, **ROW_COMM**)

MPI_BCAST(**block**, **COL_COMM**)

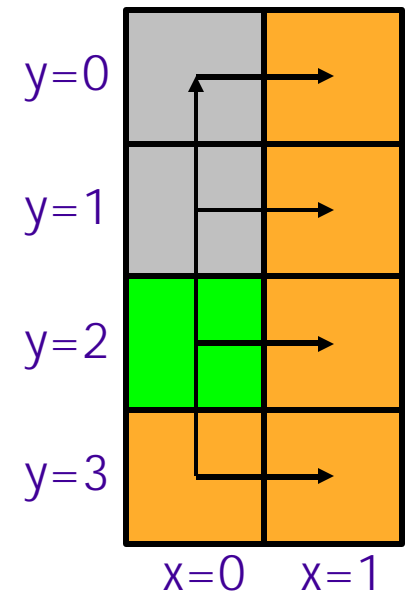
- Performing operation on a physical string of nodes along z-axis:



- ❖ A simple multicast to all nodes in z-axis will do the trick
- ❖ for a **blue block**

- Performing operation on a physical xy-plane:

- ❖ A double multicast, first along y-axis and then along x-axis
- ❖ will flood the plane with a **green block**



Summary on MPI Programming of BG/L

- Efficient mapping of operations on logical grid to real compute nodes require some forethought: interaction between job submission and run-time
- Broadcasts and reduces can be performed very efficiently in BG/L if they map to regular subsections of compute node grid
 - ❖ strings along an axis
 - ❖ planes of the three-dimensional interconnect
- We did not mention in this talk, but "where" the data is in memory is very important to BG/L
 - ❖ performance critical data should be 16-byte aligned
 - ❖ requirement for both send and received
- Goal is to simplify life of programmer, let the system do it!

Conclusions

- Embedded technology promises to be an efficient path toward building massively parallel computers optimized at the system level
 - ❖ Low power is critical to achieving a dense, simple, inexpensive system
- We are developing a BG/L system software stack with Linux-like personality for user applications
 - ❖ Custom solution (HPK) on compute nodes for highest performance
 - ❖ Linux solution on I/O nodes for flexibility and functionality
 - ❖ MPI is the default programming model, but others are being investigated
- BG/L is testing software approaches to management/operation of very large scale machines
 - ❖ Hierarchical organization for management
 - ❖ “Flat” organization for programming
 - ❖ Mixed conventional/special-purpose operating systems
- Many challenges ahead, particularly in performance and reliability

BlueGene/L Team

- NR Adiga, G Almasi, GS Almasi, Y Aridor, Leonardo Bachega, M. Bae, D Beece, R Bellofatto, G Bhanot, R Bickford, M Blumrich, AA Bright, J Brunheroto, C Cascaval, J Castaños, W Chan, L Ceze, P Coteus, S Chatterjee, D Chen, G Chiu, TM Cipolla, P Crumley, KM Desai, A Deutsch, T Domany, MB Dombrowa, W Donath, M Eleftheriou, C Erway, J Esch, B Fitch, J Gagliano, A Gara, R Garg, R Germain, ME Giampapa, B Gopalsamy, J Gunnels, M Gupta, F Gustavson, S Hall, RA Haring, D Heidel, P Heidelberger, LM Herger, D Hoenicke, RD Jackson, T Jamal-Eddine, GV Kopcsay, E Krevat, MP Kurhekar, AP Lanzetta, D Lieber, LK Liu, M Lu, M Mendell, Pedro Mindlin, A Misra, Y Moatti, L Mok, JE Moreira, BJ Nathanson, M Newton, M Ohmacht, A Oliner, V Pandit, RB Pudota, William Pulleyblank, R Rand, R Regan, B Rubin, A Ruehli, S Rus, RK Sahoo, A Sanomiya, E Schenfeld, M Sharma, E Shmueli, S Singh, P Song, V Srinivasan, BD Steinmacher-Burow, K Strauss, C Surovic, R Swetz, T Takken, RB Tremaine, M Tsao, AR Umamaheshwaran, P Verma, P Vranas, TJC Ward, M Wazlowski (IBM)
- B de Supinski, L Kissel, M Seager, J Vetter, RK Yates (LLNL)

BlueGene/L External Collaborators

- Argonne National Laboratory
- Boston University
- Caltech
- Columbia University
- National Center for Atmospheric Research
- Oak Ridge National Laboratory
- San Diego Supercomputing Center
- Stanford University
- Technical University of Vienna
- Trinity College Dublin
- Universidad Politecnica de Valencia
- Universitat Politecnica de Catalunya (Barcelona)
- University of Edinburgh
- University of Illinois at Urbana-Champaign
- University of Maryland
- University of New Mexico



IBM Research



The BlueGene/L Supercomputer: Delivering Large Scale Parallelism



José E. Moreira

IBM T. J. Watson Research Center